Sample Exam Questions
69714 - REAL TIME OPERATING SYSTEM M
MEng in Automation Engineering, University of Bologna
Academic Year 2012-2013
Paolo Torroni

This is a collection of past exam questions. Some of the questions are taken from the course text books.\(^1\) For the time being, solutions are not provided.

Be aware that “Midterm” and “Final” exams only cover part of the syllabus. A “Standard” exam instead covers all the syllabus. Exam rules and organization of the course are summarized in the first set of slides.\(^2\)

Licence

The original material in this collection is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.\(^3\) This is a Free Culture licence. You are free:

- to Share: to copy, distribute and transmit the work
- to Remix: to adapt the work
- to make commercial use of the work

Under the following conditions:

- Attribution: You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- Share Alike: If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

---


\(^2\)https://campus.unibo.it/105196

\(^3\)http://creativecommons.org/licenses/by-sa/3.0/
Please, fill in your data in the fields below. E-mail will be used for communication of the exam results.

Name ............... 
Registration No. ... 
E-mail ............. 

Part I

Quizzes (2 points)

Mark each of the following statements True or False. Explain your answer in one sentence (if you wish).

Q1) T F After a fork(), the child process and the parent process have no shared address space.
Explanation (optional):

Q2) T F RPCs use a message-based communication scheme to provide a remote service.
Explanation (optional):

Q3) T F A disadvantage of the M:M threading model is that developers cannot create as many threads as necessary, since kernel threads cannot run in parallel on a multiprocessor.
Explanation (optional):

Q4) T F Ordinary pipes continue to exist even after the processes have finished communicating and have terminated.
Explanation (optional):

Part II

Open questions (4 points)

O1) Illustrate the microkernel approach to OS design. Comment on advantages and disadvantages.

O2) Describe the differences among short-term, medium-term, and long term scheduling.

O3) There are many possible CPU-scheduling algorithms. How could we select one particular algorithm for a particular system? Discuss various evaluation methods.

Part III

Exercise (3 points)

Suppose that processes $P_1, P_2, \ldots, P_5$ arrive for execution at the times indicated in Table 1. Each process will run for the amount of time listed, and will be assigned a priority ranging from 0 (highest) to 10 (lowest). No more processes will arrive until the last process completes.

In answering the questions, base all decisions on the information you have at the time the decision must be made.

Table 1: Process arrival/CPU-burst times and priorities.

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0.0</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>$P_2$</td>
<td>0.4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>$P_3$</td>
<td>0.5</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0.8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$P_5$</td>
<td>1.0</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

E1) Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms:

E1.1) FCFS;
E1.2) preemptive SJF;
E1.3) preemptive priority (SJF if priority is equal);
E1.4) RR (quantum=2).

E2) What is the turnaround time of each process for each of these four scheduling algorithms?

E3) What is the waiting time of each process for each of these four scheduling algorithms?

E4) Which of the algorithms results in the maximum overall turnaround time (over all processes)?
Part IV
Code analysis (2 points)

The program below uses the Phreads API.

C1) What would be the output from the program at LINE A, LINE B, and LINE C?

C2) How many processes/threads would be active by the time LINE B is executed?

Justify your answers. If there are different possible answers, explain what the possibilities are.

```c
#include <sys/types.h>
#include <sys/wait.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

int value = 5;
void *runner1(void *param);
void *runner2(void *param);

int main()
{
    pthread_t tid1, tid2;
    pthread_attr_t attr1, attr2;

    pthread_attr_init(&attr1);
    pthread_create(&tid1,&attr1,runner1,NULL);
    pthread_attr_init(&attr2);
    pthread_create(&tid2,&attr2,runner2,NULL);

    printf("A: value = %d\n", value); /* LINE A */

    pthread_join(tid1,NULL);
    pthread_join(tid2,NULL);

    return 0;
}

void *runner1(void *param) {
    value += 10;
    printf("B: value = %d\n", value); /* LINE B */
    pthread_exit(0);
}

void *runner2(void *param) {
    value += 10;
    printf("C: value = %d\n", value); /* LINE C */
    pthread_exit(0);
}
```
Please, fill in your data in the fields below. E-mail will be used for communication of the exam results.

Name ............... 
Registration No. ... 
E-mail .............

Part I
Quizzes (2 points)

Mark each of the following statements True or False. Explain your answer in one sentence (if you wish).

Q1) T F After a fork(), the child process and the parent process have no open files in common.
Explanation (optional):

Q2) T F A rendezvous can be obtained using a blocking send() and a blocking receive().
Explanation (optional):

Q3) T F A disadvantage of the many-to-one threading model is that the entire process will block if a thread makes a blocking system call.
Explanation (optional):

Q4) T F A wait() system call is used to make a process wait for an input/output device to become ready.
Explanation (optional):

Part II
Open questions (4 points)

O1) What is the purpose of interrupts? Explain how interrupt-driven system operation can be obtained using dual mode operation and system calls.
O2) Explain scheduling queues.
O3) Describe the differences between direct communication and indirect communication in message-based systems.

Part III
Exercise (3 points)

Suppose that processes $P_1, P_2, \ldots, P_5$ arrive for execution at the times indicated in Table 1. Each process will run for the amount of time listed, and will be assigned a priority ranging from 0 (highest) to 10 (lowest). No more processes will arrive until the last process completes.

In answering the questions, base all decisions on the information you have at the time the decision must be made.

Table 1: Process arrival/CPU-burst times and priorities.

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0.0</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>$P_2$</td>
<td>0.4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>$P_3$</td>
<td>0.5</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0.8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$P_5$</td>
<td>1.0</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

E1) Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms:
E1.1) nonpreemptive SJF;
E1.2) preemptive SJF;
E1.3) preemptive priority (FCFS if priority is equal);
E1.4) RR (quantum=4).

E2) What is the turnaround time of each process for each of these four scheduling algorithms?
E3) What is the waiting time of each process for each of these four scheduling algorithms?
E4) Which of the algorithms results in the minimum average waiting time (over all processes)?
Part IV
Code analysis (2 points)

The program below uses the Phreads API.

C1) What would be the output from the program at LINE A, LINE B, and LINE C?

C2) How many processes/threads would be active by the time LINE B is executed?

Justify your answers. If there are different possible answers, explain what the possibilities are.

```c
#include <sys/types.h>
#include <sys/wait.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

int value = 5;
void *runner(void *param);

int main()
{
    pid_t pid;
    pthread_t tid;
    pthread_attr_t attr;

    printf("A: value = %d\n", value); /* LINE A */
    pid = fork();
    if (pid == 0) {
        pthread_attr_init(&attr);
        pthread_create(&tid,&attr,runner,NULL);
        pthread_join(tid,NULL);
        printf("B: value = %d\n", value); /* LINE B */
        return 0;
    }
    else if (pid > 0) {
        wait(NULL);
        printf("C: value = %d\n", value); /* LINE C */
        return 0;
    }
    return 0;
}

void *runner(void *param) {
    value += 10;
    pthread_exit(0);
}
```
Real-Time Operating Systems M  
Second Midterm. May 13, 2013

Please, fill in your data in the fields below. E-mail will be used for communication of the exam results.

Name ………………
Registration No. …
E-mail …………..

Part I  
Quizzes (2 points)

Mark each of the following statements True or False. Explain your answer in one sentence (if you wish).

Q1) T F A process cannot be executed if its logical address space is bigger than the size of the physical memory.
Explanation (optional):

Q2) T F An inverted page table contains, in each entry, a page number and a frame number.
Explanation (optional):

Q3) T F Paging eliminates internal fragmentation.
Explanation (optional):

Q4) T F Deadlock cannot occur among processes that need at most one (non-shareable) resource each.
Explanation (optional):

Part II  
Open questions (3 points)

O1) Why do some operating systems use spinlocks as a synchronisation mechanism only on multiprocessor systems and not on single-processor systems?

O2) What is the cause of thrashing? How does the system detect thrashing? Once it detects thrashing, what can the system do to eliminate the problem?

O3) Explain how data can be transferred from a device to the main memory using a direct memory access (DMA) controller.

Part III  
Exercises (3 points)

E1) Consider the following snapshot of a system:

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td>A B C D</td>
<td>A B C D</td>
</tr>
<tr>
<td>P_1</td>
<td>0 0 1 2</td>
<td>0 0 2 3</td>
</tr>
<tr>
<td>P_2</td>
<td>1 0 0 0</td>
<td>1 2 2 0</td>
</tr>
<tr>
<td>P_3</td>
<td>1 3 5 4</td>
<td>2 3 5 6</td>
</tr>
<tr>
<td>P_4</td>
<td>0 0 0 1</td>
<td>2 2 0 1</td>
</tr>
<tr>
<td>Available</td>
<td>1 2 2 0</td>
<td></td>
</tr>
</tbody>
</table>

E1.1) Is the system in deadlock?
E1.2) Is the system in a safe state?
E1.3) Can P_3’s request be safely granted immediately?
E1.4) If P_3’s request is granted immediately, does the system enter a deadlock?

Be sure to motivate your answers.

E2) Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of 210 KB, 420 KB, 110 KB, and 430 KB (in order)? Which algorithm makes the most efficient use of memory?

E3) Consider the following reference string:

1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 6, 5, 7.

Suppose you have four page frames. How may page faults occur for the LRU page replacement algorithm? Is that the minimum possible number of page faults?

Part IV  
Code analysis (3 points)

The Cigarette-Smokers Problem is a well-known process synchronisation problem that can be defined as follows:
Consider a system with three smoker processes. Each smoker continuously rolls a cigarette (`make_cigarette()`) and then smokes it (`smoke()`). But to roll a cigarette, the smoker needs three resources: tobacco, paper, and matches. One of the smokers has an infinite amount of paper, another has infinite tobacco, and the third has infinite matches. In order to provide a given smoker with the missing two resources, so he can roll a cigarette, a number of other processes synchronise with one another and with the smokers. At one time, only one smoker can acquire the missing two resources. Smokers cannot accumulate resources for future use. Several smokers can smoke at the same time, but each can smoke at most one cigarette at a time.

The following pseudo-code shows a possible solution with nine processes: three smokers, three pushers, and three agents. All nine processes execute concurrently.

A1) Show a possible execution that reaches LINE A. Be sure to indicate which instructions are executed by which process before LINE A is executed.

A2) Is the mutex semaphore needed? What happens if we remove all occurrences of `wait(mutex)` and `signal(mutex)` from this solution?

A3) Show how other possible synchronisation issues—in particular, deadlock and starvation—are solved (or show what synchronisation issues are unsolved, if any).

```c
/* semaphores and shared global variables */
semaphore agentSem = 1, mutex = 1;
semaphore tobacco = 0, paper = 0, match = 0;
semaphore paper_and_matches = 0,
tobacco_and_matches = 0,
tobacco_and_paper = 0;
Boolean isPaper = FALSE, isTobacco = FALSE,
isMatches = FALSE;

/* smoker with tobacco */
while(TRUE) {
    wait(paper_and_matches);
    make_cigarette();
    signal(agentSem);
    smoke(); /* LINE A */
}

/* smoker with paper */
while(TRUE) {
    wait(tobacco_and_matches);
    make_cigarette();
    signal(agentSem);
    smoke();
}

/* smoker with matches */
while(TRUE) {
    wait(tobacco_and_paper);
    make_cigarette();
    signal(agentSem);
    smoke();
}

/* tobacco agent */
while(TRUE) {
    wait(agentSem);
    signal(paper);
    signal(matches);
}

/* paper agent */
while(TRUE) {
    wait(agentSem);
    signal(tobacco);
    signal(matches);
}

/* matches agent */
while(TRUE) {
    wait(agentSem);
    signal(tobacco);
    signal(paper);
}

/* tobacco pusher */
while(TRUE) {
    wait(tobacco);
    wait(mutex);
    if(isPaper) {
        isPaper = FALSE;
        signal(tobacco_and_paper);
    } else if(isMatches) {
        isMatches = FALSE;
        signal(tobacco_and_matches);
    } else isTobacco = TRUE;
    signal(mutex);
}

/* paper pusher */
while(TRUE) {
    wait(paper);
    wait(mutex);
    if(isTobacco) {
        isTobacco = FALSE;
        signal(tobacco_and_paper);
    } else if(isMatches) {
        isMatches = FALSE;
        signal(paper_and_matches);
    } else isPaper = TRUE;
    signal(mutex);
}

/* matches pusher */
while(TRUE) {
    wait(matches);
    wait(mutex);
    if(isPaper) {
        isPaper = FALSE;
        signal(paper_and_matches);
    } else if(isTobacco) {
        isTobacco = FALSE;
        signal(tobacco_and_matches);
    } else isMatches = TRUE;
    signal(mutex);
}
Real-Time Operating Systems M
Second Midterm. May 13, 2013

Part I
Quizzes (2 points)
Mark each of the following statements True or False. Explain your answer in one sentence (if you wish).

Q1) \( \text{T} \quad \text{F} \) A hashed page table contains, in each entry, a pointer to a list.

Explanation (optional):

Q2) \( \text{T} \quad \text{F} \) Deadlock cannot occur among processes that request (non-shareable) resources only when they have none.

Explanation (optional):

Q3) \( \text{T} \quad \text{F} \) Paging permits pages to be of arbitrary size.

Explanation (optional):

Q4) \( \text{T} \quad \text{F} \) The working-set model is used to prevent thrashing while at the same time optimising CPU utilisation.

Explanation (optional):

Part II
Open questions (3 points)

O1) How does the signal() operation on condition variables associated with monitors differ from the signal() operation defined for semaphores?

Part III
Exercises (3 points)

E1) Consider the following snapshot of a system:

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td>A B C D</td>
<td>A B C D</td>
</tr>
<tr>
<td>P1</td>
<td>0 0 1 2</td>
<td>0 0 2 3</td>
</tr>
<tr>
<td>P2</td>
<td>1 3 5 4</td>
<td>2 3 5 6</td>
</tr>
<tr>
<td>P3</td>
<td>1 0 0 0</td>
<td>1 2 2 0</td>
</tr>
<tr>
<td>P4</td>
<td>0 0 0 1</td>
<td>2 2 0 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 2 0</td>
</tr>
</tbody>
</table>

E1.1) Is the system in deadlock?
E1.2) Is the system in a safe state?
E1.3) Can \( P_2 \)'s request be safely granted immediately?
E1.4) If \( P_2 \)'s request is granted immediately, does the system enter a deadlock?

Be sure to motivate your answers.

E2) Consider a logical address space of 32 pages with 1,024 words per page, mapped onto a physical memory of 16 frames. How many bits are required in the logical address? How many bits are required in the physical address?

E3) Consider the following reference string:

4, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 9, 7, 8, 9, 5, 6, 5, 7, 5, 6.

Suppose you have four page frames. How many page faults occur for the LRU page replacement algorithm? Is that the minimum possible number of page faults?

Part IV
Code analysis (3 points)

The Cigarette-Smokers Problem is a well-known process synchronisation problem that can defined as follows:
Consider a system with three smoker processes. Each smoker continuously rolls a cigarette (make_cigarette()) and then smokes it (smoke()). But to roll a cigarette, the smoker needs three resources: tobacco, paper, and matches. One of the smokers has an infinite amount of paper, another has infinite tobacco, and the third has infinite matches.

In order to provide a given smoker with the missing two resources, so he can roll a cigarette, a number of other processes synchronise with one another and with the smokers.

At one time, only one smoker can acquire the missing two resources. Smokers cannot accumulate resources for future use. Several smokers can smoke at the same time, but each can smoke at most one cigarette at a time.

The following pseudo-code shows a possible solution with nine processes: three smokers, three pushers, and three agents. All nine processes execute concurrently.

A1) Show a possible execution that reaches LINE A. Be sure to indicate which instructions are executed by which process before LINE A is executed.

A2) Is the mutex semaphore needed? What happens if we remove all occurrences of wait(mutex) and signal(mutex) from this solution?

A3) Show how other possible synchronisation issues—in particular, deadlock and starvation—are solved (or show what synchronisation issues are unsolved, if any).

```c
/* semaphores and shared global variables */
semaphore agentSem = 1, mutex = 1;
semaphore tobacco = 0, paper = 0, match = 0;
semaphore paper_and_matches = 0, tobacco_and_matches = 0,
tobacco_and_paper = 0;
Boolean isPaper = FALSE, isTobacco = FALSE,
isMatches = FALSE;

/* smoker with tobacco */
while(TRUE) {
    wait(paper_and_matches);
    make_cigarette();
    signal(agentSem);
    smoke();
}

/* smoker with paper */
while(TRUE) {
    wait(tobacco_and_matches);
    make_cigarette();
    signal(agentSem);
   -smoke(); /* LINE A */
}

/* smoker with matches */
while(TRUE) {
    wait(tobacco_and_paper);
    make_cigarette();
    signal(agentSem);
    smoke();
}

/* tobacco agent */
while(TRUE) {
    wait(agentSem);
    signal(paper);
    signal(matches);
}

/* paper agent */
while(TRUE) {
    wait(agentSem);
    signal(tobacco);
    signal(matches);
}

/* matches agent */
while(TRUE) {
    wait(agentSem);
    signal(tobacco);
    signal(paper);
}

/* tobacco pusher */
while(TRUE) {
    wait(tobacco);
    wait(mutex);
    if(isPaper) {
        isPaper = FALSE;
        signal(tobacco_and_paper);
    } else if(isMatches) {
        isMatches = FALSE;
        signal(tobacco_and_matches);
    } else isTobacco = TRUE;
    signal(mutex);
}

/* paper pusher */
while(TRUE) {
    wait(paper);
    wait(mutex);
    if(isTobacco) {
        isTobacco = FALSE;
        signal(tobacco_and_paper);
    } else if(isMatches) {
        isMatches = FALSE;
        signal(paper_and_matches);
    } else isPaper = TRUE;
    signal(mutex);
}

/* matches pusher */
while(TRUE) {
    wait(matches);
    wait(mutex);
    if(isPaper) {
        isPaper = FALSE;
        signal(paper_and_matches);
    } else if(isTobacco) {
        isTobacco = FALSE;
        signal(tobacco_and_matches);
    } else isMatches = TRUE;
    signal(mutex);
}
Real-Time Operating Systems M
First+Second Midterm. May 13, 2013

Please, fill in your data in the fields below. E-mail will be used for communication of the exam results.

Name ............
Registration No. ...
E-mail ............

Part I
Quizzes (4 points)

Mark each of the following statements True or False. Explain your answer in one sentence (if you wish).

Q1) T F A **rendezvous** can be obtained using a blocking **send()** and a blocking **receive()**.

*Explanation (optional):*

Q2) T F A disadvantage of the **M:M threading model** is that developers cannot create as many threads as necessary, since kernel threads cannot run in parallel on a multiprocessor.

*Explanation (optional):*

Q3) T F Immediately after a **fork()** is executed, the child process and the parent process have **no shared variables**.

*Explanation (optional):*

Q4) T F **Named pipes** continue to exist even after the processes have finished communicating and have terminated.

*Explanation (optional):*

Q5) T F Paging eliminates **internal fragmentation**.

*Explanation (optional):*

Q6) T F **Deadlock** cannot occur among processes that need at most one (non-shareable) resource each.

*Explanation (optional):*

Q7) T F The **working-set model** is used to prevent thrashing while at the same time optimising CPU utilisation.

*Explanation (optional):*

Q8) T F A **hashed page table** contains, in each entry, a pointer to a list.

*Explanation (optional):*

Part II
Open questions (7 points)

O1) Illustrate the **modular kernel** approach to OS design. Comment on advantages and disadvantages.

O2) There are many possible CPU-scheduling algorithms. How could we select one particular algorithm for a particular system? Discuss various **evaluation methods**.

O3) Under what circumstances do **page faults** occur? Describe the actions taken from the operating system when a page fault occurs.

O4) Explain how data can be transferred from a device to the main memory using a **direct memory access (DMA) controller**.

O5) Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Show that the system is **deadlock free**.
Part III
Exercises (6 points)

E1) Suppose that processes $P_1, P_2, \ldots, P_5$ arrive for execution at the times indicated in Table 1. Each process will run for the amount of time listed, and will be assigned a priority ranging from 0 (highest) to 10 (lowest). No more processes will arrive until the last process completes.

Table 1: Process arrival/CPU-burst times and priorities.

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0.0</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>$P_2$</td>
<td>0.4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>$P_3$</td>
<td>0.5</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0.8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$P_5$</td>
<td>1.0</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms:

E1.1) nonpreemptive SJF;
E1.2) preemptive SJF;
E1.3) preemptive priority (FCFS if priority is equal);
E1.4) RR (quantum=4).

In answering the questions, base all decisions on the information you have at the time the decision must be made.

E2) Consider the following snapshot of a system:

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td>A B C D</td>
</tr>
<tr>
<td>$P_1$</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>$P_2$</td>
<td>0 0 1 2</td>
</tr>
<tr>
<td>$P_3$</td>
<td>1 3 5 4</td>
</tr>
<tr>
<td>$P_4$</td>
<td>1 0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 2 0</td>
</tr>
</tbody>
</table>

E1.1) Is the system in deadlock?
E1.2) Is the system in a safe state?
E1.3) Can $P_3$’s request be safely granted immediately?
E1.4) If $P_3$’s request is granted immediately, does the system enter a deadlock?

Be sure to motivate your answers.

E3) Consider the following reference string:

1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2.

Suppose you have four page frames. How may page faults occur for the LRU page replacement algorithm? Is that the minimum possible number of page faults?

Part IV
Code analysis (5 points)

C1) The program below uses the Phreads API.

C1.1) What would be the output from the program at LINE A, LINE B, and LINE C?
C1.2) How many processes/threads would be active by the time LINE B is executed?

Be sure to justify your answers.

```c
#include <sys/types.h>
#include <sys/wait.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

int val = 10;

void *runner(void *param) {
    val += 20;
    pthread_exit(0);
}

int main() {
    pid_t pid;
    pthread_t tid;
    pthread_attr_t attr;
    printf("A: value = %d\n", val); /* LINE A */
    pid = fork();
    if (pid == 0) {
        pthread_attr_init(&attr);
        pthread_create(&tid,&attr,runner,NULL);
        pthread_join(tid,NULL);
        printf("B: value = %d\n", val); /* LINE B */
        return 0;
    }
    else if (pid > 0) {
        wait(NULL);
        printf("C: value = %d\n", val); /* LINE C */
        return 0;
    }
    return 0;
}
```

void *runner(void *param) {
    val += 20;
    pthread_exit(0);
}

C2) The Cigarette-Smokers Problem is a well-known process synchronisation problem, defined as follows:

Consider a system with three smoker processes. Each smoker continuously rolls a cigarette (make_cigarette()) and then smokes it (smoke()). But to roll a cigarette, the smoker needs three resources: tobacco, paper, and matches. One of the smokers has an infinite amount of paper, another has infinite tobacco, and the third has infinite matches.

In order to provide a given smoker with the missing two resources, so he can roll
a cigarette, a number of other processes synchronise with one another and with the 
smokers.
At one time, only one smoker can acquire 
the missing two resources. Smokers cannot 
accumulate resources for future use. Several 
smokers can smoke at the same time, but 
each can smoke at most one cigarette at a 
time.

The following pseudo-code shows a possible solution 
with nine processes: three smokers, three pushers, and 
three agents. All nine processes execute concurrently.

C2.1) Show a possible execution that reaches LINE A. 
Be sure to indicate which instructions are 
executed by which process before LINE A is 
executed.
C2.2) Is the mutex semaphore needed? What happens 
if we remove all occurrences of wait(mutex) 
and signal(mutex) from this solution?
C2.3) Show how other possible synchronisation issues– 
in particular, deadlock and starvation–are 
solved (or show what synchronisation issues are 
unsolved, if any).

```c
/* semaphores and shared global variables */
semaphore agentSem = 1, mutex = 1;
semaphore tobacco = 0, paper = 0, match = 0;
semaphore tobacco_and_matches = 0,
  tobacco_and_paper = 0,
semaphore paper_and_matches = 0,
Boolean isPaper = FALSE, isTobacco = FALSE,
isMatches = FALSE;

/* smoker with tobacco */
while(TRUE) {
  wait(paper_and_matches);
  make_cigarette();
  signal(agentSem);
  smoke();
}

/* smoker with paper */
while(TRUE) {
  wait(tobacco_and_matches);
  make_cigarette();
  signal(agentSem);
  smoke();
}

/* smoker with matches */
while(TRUE) {
  wait(tobacco_and_paper);
  make_cigarette();
  signal(agentSem);
  smoke(); /* LINE A */
}

/* tobacco agent */
while(TRUE) {
  wait(agentSem);
  signal(paper);
  signal(matches);
}

/* paper agent */
while(TRUE) {
  wait(agentSem);
  signal(tobacco);
  signal(matches);
}

/* matches agent */
while(TRUE) {
  wait(agentSem);
  signal(tobacco);
  signal(paper);
}

/* tobacco pusher */
while(TRUE) {
  wait(tobacco);
  wait(mutex);
  if(isPaper) {
    isPaper = FALSE;
    signal(tobacco_and_paper);
  } else if(isMatches) {
    isMatches = FALSE;
    signal(tobacco_and_matches);
  } else isTobacco = TRUE;
  signal(mutex);
}

/* paper pusher */
while(TRUE) {
  wait(paper);
  wait(mutex);
  if(isTobacco) {
    isTobacco = FALSE;
    signal(tobacco_and_paper);
  } else if(isMatches) {
    isMatches = FALSE;
    signal(paper_and_matches);
  } else isPaper = TRUE;
  signal(mutex);
}

/* matches pusher */
while(TRUE) {
  wait(matches);
  wait(mutex);
  if(isPaper) {
    isPaper = FALSE;
    signal(paper_and_matches);
  } else if(isTobacco) {
    isTobacco = FALSE;
    signal(tobacco_and_matches);
  } else isMatches = TRUE;
  signal(mutex);
}
```
Please, fill in your data in the fields below. E-mail will be used for communication of the exam results.

Name ...............  
Registration No. ...  
E-mail ...............

Frequently used formulas and tables
Some of the following formulas and tables may be useful in solving some of the exercises.

Schedulability Analysis (Extended)

\[ \forall i = 1, \ldots, n \sum_{h:P_h > P_i} \frac{C_h}{T_h} + \frac{C_i + B_i}{T_i} \leq i(2^{1/i} - 1) \]

\[ \forall i = 1, \ldots, n \prod_{h:P_h > P_i} \left( \frac{C_h}{T_h} + 1 \right) \left( \frac{C_i + B_i}{T_i} + 1 \right) \leq 2 \]

\[ \forall i = 1, \ldots, n \sum_{h:P_h > P_i} \frac{C_h}{T_h} + \frac{C_i + B_i}{T_i} \leq 1 \]

Response Time Analysis (Extended)

\[
\begin{align*}
R_i^{(0)} & = C_i + B_i + \sum_{k=1}^{i-1} C_k \\
R_i^{(s)} & = C_i + B_i + I_i^{(s-1)} = C_i + B_i + \sum_{k=1}^{i-1} \left[ \frac{R_i^{(s-1)}}{T_k} \right] C_k 
\end{align*}
\]

Processor Demand Test

\[ g(0, L) = \sum_{i=1}^{n} \left[ \frac{L - D_i + T_i}{T_i} \right] C_i \]

\[ L^* = \frac{1}{1 - U} \sum_{i=1}^{n} (T_i - D_i) U_i \]

<table>
<thead>
<tr>
<th>n</th>
<th>n(2^{1/n} - 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.000</td>
</tr>
<tr>
<td>2</td>
<td>0.828</td>
</tr>
<tr>
<td>3</td>
<td>0.780</td>
</tr>
<tr>
<td>4</td>
<td>0.757</td>
</tr>
<tr>
<td>5</td>
<td>0.743</td>
</tr>
<tr>
<td>6</td>
<td>0.735</td>
</tr>
<tr>
<td>7</td>
<td>0.729</td>
</tr>
<tr>
<td>8</td>
<td>0.724</td>
</tr>
<tr>
<td>9</td>
<td>0.721</td>
</tr>
<tr>
<td>10</td>
<td>0.718</td>
</tr>
</tbody>
</table>

Part I
Quizzes (2 points)

Mark each of the following statements True or False. Explain your answer in one sentence (if you wish).

Q1) T F The Stack Resource Policy may stop a task allowed by the Non-Preemptive Protocol.

Explanation (optional):

Q2) T F Given a set of independent aperiodic tasks, with arbitrary arrival times, any algorithm that executes the tasks in order of nondecreasing relative deadlines is optimal with respect to minimising the maximum lateness.

Explanation (optional):

Q3) T F Push-through blocking cannot affect a highest-priority task.

Explanation (optional):

Q4) T F For independent preemptive periodic tasks under fixed priorities, the critical instant of a given task occurs when all higher priority tasks have all different activation times.

Explanation (optional):

Q5) T F The processor utilization’s least upper bound $U_{ub}$ distinguishes between feasible and infeasible task sets.

Explanation (optional):

Q6) T F EDF is a simpler but more rigid scheduling algorithm than Timeline scheduling.

Explanation (optional):
Part II
Open questions (3 points)

O1) Discuss the main properties of the Stack Resource Policy protocol.

O2) Compare advantages and disadvantages of complete tree-search algorithms (such as Bratley’s) with respect to heuristic algorithms (such as the one used in the Spring kernel) for real-time task scheduling.

Part III
Exercises (6 points)

E1) Let $a_i$, worst-case computation time $C_i$, absolute deadline $d_i$, relative to $a_i$, and precedence relations of each task $\tau_i$ in $\Gamma_1$.

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$C_i$</th>
<th>$d_i$</th>
<th>$\text{prec}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>20</td>
<td>$\tau_2 \rightarrow \tau_4$</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>20</td>
<td>$\tau_1 \rightarrow \tau_5$</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>19</td>
<td>$\tau_3 \rightarrow \tau_4$</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>18</td>
<td>$\tau_4 \rightarrow \tau_6$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>20</td>
<td>$\tau_5 \rightarrow \tau_6$</td>
</tr>
</tbody>
</table>

Figure 1: Characteristics of the $\Gamma_1$ task set.

E2) Consider a set of periodic tasks $\Gamma_2 = \tau_1, \ldots, \tau_4$, to be scheduled on a single-processor machine. Figure 2 shows period $T_i$ and worst-case computation time $C_i$ of each task $\tau_i$.

<table>
<thead>
<tr>
<th>$\Gamma_2$</th>
<th>$T_i$</th>
<th>$C_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>12</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 2: Characteristics of the $\Gamma_2$ task set.

E2.1) Is $\Gamma_2$ feasible under fixed priorities?
E2.2) Is $\Gamma_2$ feasible under dynamic priorities?

Let us now assume the following relative deadlines for $\tau_2, \tau_3, \tau_4$: $D_2 = 2, D_3 = 5, D_4 = 10$, whereas $T_1 = D_1 = 3$. Let us call $\Gamma'_2$ this new task set (see Figure 3).

<table>
<thead>
<tr>
<th>$\Gamma'_2$</th>
<th>$T_i$</th>
<th>$C_i$</th>
<th>$D_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>6</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>12</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure 3: Characteristics of the $\Gamma'_2$ task set.

E2.3) Is $\Gamma'_2$ feasible under fixed priorities?
E2.4) Is $\Gamma'_2$ feasible under dynamic priorities?

Be sure to motivate your answer.

E3) Consider a task set $\Gamma_3$, composed of 5 periodic tasks $\tau_1, \ldots, \tau_5$ that share 4 resources $a, b, c, d$ and execute on a single-processor machine. $\Gamma_3$’s tasks are represented in Figure 4. Each resource is accessed in mutual exclusion using the Priority Ceiling Protocol (PCP).

Figure 4: Graphical representation of critical sections in $\Gamma_3$.

Figure 5 shows phase $\Phi_i$, period $T_i = D_i$, worst-case computation time $C_i$, and a description of the access windows to the shared resources of each task $\tau_i$, in terms of start time $t(R_k)$ and duration $\delta_{i,k}$. Below the Gantt chart, show how $\tau_5$’s active priority $p_5$ evolves.

<table>
<thead>
<tr>
<th>$\Gamma_3$</th>
<th>$\Phi_i$</th>
<th>$T_i$</th>
<th>$C_i$</th>
<th>$t(a)$</th>
<th>$\delta_{1,a}$</th>
<th>$t(b)$</th>
<th>$\delta_{1,b}$</th>
<th>$t(c)$</th>
<th>$\delta_{1,c}$</th>
<th>$t(d)$</th>
<th>$\delta_{1,d}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>8</td>
<td>20</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>6</td>
<td>30</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>4</td>
<td>40</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>2</td>
<td>50</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau_5$</td>
<td>0</td>
<td>60</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5: Characteristics of the $\Gamma_3$ task set.

E3.1) Using a Gantt chart, show the schedule under RM+PCP, from time 0 until completion of the first instance of $\tau_5$. Below the Gantt chart, show how $\tau_5$’s active priority $p_5$ evolves.
Real-Time Operating Systems M
Final Exam. June 12, 2013

Please, fill in your data in the fields below. E-mail will be used for communication of the exam results.

Name ...............
Registration No. ...
E-mail .............

Frequently used formulas and tables
Some of the following formulas and tables may be useful in solving some of the exercises.

Schedulability Analysis (Extended)
\[ \forall i = 1, \ldots, n \sum_{h: P_h > P_i} \frac{C_h}{T_h} + \frac{C_i + B_i}{T_i} \leq i(2^{1/i} - 1) \]
\[ \forall i = 1, \ldots, n \prod_{h: P_h > P_i} \left( \frac{C_h}{T_h} + 1 \right) \left( \frac{C_i + B_i}{T_i} + 1 \right) \leq 2 \]
\[ \forall i = 1, \ldots, n \sum_{h: P_h > P_i} \frac{C_h}{T_h} + \frac{C_i + B_i}{T_i} \leq 1 \]

Response Time Analysis (Extended)
\[ R_i^{(0)} = C_i + B_i + \sum_{k=1}^{i-1} C_k \]
\[ R_i^{(s)} = C_i + B_i + I_i^{(s-1)} = C_i + B_i + \sum_{k=1}^{i-1} \left[ \frac{R_k^{(s-1)}}{T_k} \right] C_k \]

Processor Demand Test
\[ g(0, L) = \sum_{i=1}^{n} \left[ \frac{L - D_i + T_i}{T_i} \right] C_i \]
\[ L^* = \frac{1}{1 - U} \sum_{i=1}^{n} (T_i - D_i) U_i \]

<table>
<thead>
<tr>
<th>n</th>
<th>n(2^{1/n} - 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.000</td>
</tr>
<tr>
<td>2</td>
<td>0.828</td>
</tr>
<tr>
<td>3</td>
<td>0.780</td>
</tr>
<tr>
<td>4</td>
<td>0.757</td>
</tr>
<tr>
<td>5</td>
<td>0.743</td>
</tr>
<tr>
<td>6</td>
<td>0.735</td>
</tr>
<tr>
<td>7</td>
<td>0.729</td>
</tr>
<tr>
<td>8</td>
<td>0.724</td>
</tr>
<tr>
<td>9</td>
<td>0.721</td>
</tr>
<tr>
<td>10</td>
<td>0.718</td>
</tr>
</tbody>
</table>

Part I
Quizzes (2 points)
Mark each of the following statements True or False. Explain your answer in one sentence (if you wish).

Q1) Under the Priority Ceiling Protocol, a task can be blocked only before it starts executing, never once it has started.
   Explanation (optional):
   T

Q2) For each task set, there exists always one and only one optimal scheduling algorithm (in the sense of feasibility).
   Explanation (optional):
   F

Q3) Preemption generally does not increase the complexity of a scheduling problem.
   Explanation (optional):
   T

Q4) For independent preemptive periodic tasks under fixed priorities, the critical instant of a given task occurs when all higher priority tasks have the same activation time as its own.
   Explanation (optional):
   F

Q5) A task set consisting of three tasks, τ1, τ2, and τ3, with identical period, is RM-feasible if and only if the total processor utilisation is at most 1.
   Explanation (optional):
   T

Q6) Priority Inversion is a phenomenon that cannot occur if tasks are independent.
   Explanation (optional):
   F
Part II

Open questions (3 points)

O1) What are the main unsolved problems of the Priority Inheritance Protocol? Use examples to illustrate the point.

O2) Compare advantages and disadvantages of RM with respect to EDF.

Part III

Exercises (6 points)

E1) Let $\Gamma_1 = \tau_1, \ldots, \tau_6$ be a set of preemptable, aperiodic tasks with precedence constraints, to be executed on a single-processor machine. Figure 1 shows release time $a_i$, worst-case computation time $C_i$, absolute deadline $d_i$, relative to $a_i$, and precedence relations of each task $\tau_i$ in $\Gamma_1$.

$$
\begin{array}{c|cccccc}
\tau_i & \tau_1 & \tau_2 & \tau_3 & \tau_4 & \tau_5 & \tau_6 \\
\hline
a_i & 0 & 6 & 4 & 13 & 0 & 10 \\
C_i & 5 & 4 & 3 & 4 & 1 & 1 \\
d_i & 20 & 20 & 19 & 18 & 20 & 18 \\
\text{prec} & \tau_2 \rightarrow \tau_4 & \tau_1 \rightarrow \tau_5 & \tau_4 \rightarrow \tau_6 & \tau_3 \rightarrow \tau_4 & \tau_5 \rightarrow \tau_6 \\
\end{array}
$$

Figure 1: Characteristics of the $\Gamma_1$ task set.

Show the minimum lateness schedule for $\Gamma_1$, using a Gantt chart. Be sure to motivate your answer.

E2) Consider a set of periodic tasks $\Gamma_2 = \tau_1, \ldots, \tau_4$, to be scheduled on a single-processor machine. Figure 2 shows period $T_i$ and worst-case computation time $C_i$ of each task $\tau_i$.

$$
\begin{array}{c|cc}
\tau_i & T_i & C_i \\
\hline
\tau_1 & 3 & 1 \\
\tau_2 & 4 & 1 \\
\tau_3 & 6 & 1 \\
\tau_4 & 12 & 2 \\
\end{array}
$$

Figure 2: Characteristics of the $\Gamma_2$ task set.

E2.1) Is $\Gamma_2$ feasible under fixed priorities?
E2.2) Is $\Gamma_2$ feasible under dynamic priorities?

Let us now assume the following relative deadlines for $\tau_2, \tau_3, \tau_4$: $D_2 = 2, D_3 = 5, D_4 = 10$, whereas $T_1 = D_1 = 3$. Let us call $\Gamma'_2$ this new task set (see Figure 3).

E2.3) Is $\Gamma'_2$ feasible under fixed priorities?
E2.4) Is $\Gamma'_2$ feasible under dynamic priorities?

Be sure to motivate your answer.

E3) Consider a task set $\Gamma_3$, composed of 5 periodic tasks $\tau_1, \ldots, \tau_5$ that share 4 resources $a, b, c, d$ and execute on a single-processor machine. $\Gamma_3$’s tasks are represented in Figure 4. Each resource is accessed in mutual exclusion using the Stack Resource Policy (SRP).

$$
\begin{array}{c|cccc}
\tau_i & T_i & C_i & t(a) & t(b) & t(c) \\
\hline
\tau_1 & 8 & 20 & 5 & 1 & 3 \\
\tau_2 & 6 & 30 & 6 & 1 & 4 \\
\tau_3 & 4 & 40 & 6 & 1 & 5 \\
\tau_4 & 2 & 50 & 3 & 1 & 1 \\
\tau_5 & 0 & 60 & 6 & 1 & 4 \end{array}
$$

Figure 3: Characteristics of the $\Gamma'_2$ task set.

Figure 4: Graphical representation of critical sections in $\Gamma_3$.

Figure 5 shows phase $\Phi$, period $T_i = D_i$, worst-case computation time $C_i$, and a description of the access windows to the shared resources of each task $\tau_i$, in terms of start time $t(R_k)$ and duration $\delta_{i,k}$ of the critical section for each task $\tau_i$ and each resource $R_k$.

E3.1) Using a Gantt chart, show the schedule under EDF+SRP, from time 0 until completion of the first instance of $\tau_5$. Below the Gantt chart, show how the system ceiling $\Pi_s$ evolves.
Real-Time Operating Systems M
Standard Exam. June 12, 2013

Please, fill in your data in the fields below. E-mail will be used for communication of the exam results.

Name .................
Registration No. 
E-mail .................

Frequently used formulas and tables
Some of the following formulas and tables may be useful in solving some of the exercises.

Schedulability Analysis (Extended)
\[ \forall i = 1, \ldots, n \sum_{h: p_h > P_i} \frac{C_h}{T_h} + \frac{C_i + B_i}{T_i} \leq i(2^{\frac{i}{n}} - 1) \]

\[ \forall i = 1, \ldots, n \prod_{h: p_h > P_i} \left( \frac{C_h}{T_h} + 1 \right) \left( \frac{C_i + B_i}{T_i} + 1 \right) \leq 2 \]

\[ \forall i = 1, \ldots, n \sum_{h: p_h > P_i} \frac{C_h}{T_h} + \frac{C_i + B_i}{T_i} \leq 1 \]

Response Time Analysis (Extended)
\[
\begin{align*}
R_i^{(0)} &= C_i + B_i + \sum_{k=1}^{i-1} C_k \\
R_i^{(s)} &= C_i + B_i + I_i^{(s-1)} = C_i + B_i + \sum_{k=1}^{i-1} \left[ \frac{R_i^{(s-1)}}{T_k} \right] C_k 
\end{align*}
\]

Processor Demand Test
\[
g(0, L) = \sum_{i=1}^{n} \left[ \frac{L - D_i + T_i}{T_i} \right] C_i \\
L^* = \frac{1}{1 - U} \sum_{i=1}^{n} (T_i - D_i) U_i
\]

<table>
<thead>
<tr>
<th>n</th>
<th>n(2^{\frac{i}{n}} - 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.000</td>
</tr>
<tr>
<td>2</td>
<td>0.828</td>
</tr>
<tr>
<td>3</td>
<td>0.780</td>
</tr>
<tr>
<td>4</td>
<td>0.757</td>
</tr>
<tr>
<td>5</td>
<td>0.743</td>
</tr>
<tr>
<td>6</td>
<td>0.735</td>
</tr>
<tr>
<td>7</td>
<td>0.729</td>
</tr>
<tr>
<td>8</td>
<td>0.724</td>
</tr>
<tr>
<td>9</td>
<td>0.721</td>
</tr>
<tr>
<td>10</td>
<td>0.718</td>
</tr>
</tbody>
</table>

Part I
Quizzes (8 pts)

Mark each of the following statements True or False. Explain your answer in one sentence (if you wish).

Q1) In the indirect communication IPC scheme, a communication link may be associated with at most two processes.

Explanation (optional):

Q2) An advantage of multithreading is an increased responsiveness in interactive applications.

Explanation (optional):

Q3) All named pipes created by a process P are automatically removed from the file system after P terminates.

Explanation (optional):

Q4) Paging eliminates external fragmentation.

Explanation (optional):

Q5) Deadlock cannot occur among processes that need at most two (non-shareable) resources each.

Explanation (optional):

Q6) The working-set model is used to prevent thrashing while at the same time optimising CPU utilisation.

Explanation (optional):

Q7) For independent preemptive periodic tasks with fixed priorities, the critical instant of a given task occurs when all higher priority tasks have the same activation time as its own.

Explanation (optional):
Q11) T Explain why the Priority Ceiling Protocol prevents deadlock.

Q12) T Under the Priority Ceiling Protocol, a task can be blocked only before it starts executing, never once it has started.

Explanation (optional):

Part II

Open questions (10 pts)

O1) Describe the content of a Process Control Block.

O2) Illustrate the Dining Philosophers Problem. Show a possible solution using semaphores. Discuss three different ways to prevent deadlock.

O3) What are the Priority Inheritance Protocol’s main unsolved problems? Use examples to illustrate the point.

Part III

Exercises (13 pts)

E1) Consider the following reference string:

![Reference String](image)

Suppose you have four page frames. How many page faults occur for the LRU page replacement algorithm? Is that the minimum possible number of page faults? Be sure to motivate your answer.

E2) Let \( \Gamma_2 = \tau_1, \ldots, \tau_6 \) be a set of non-preemptable, aperiodic and synchronous tasks with precedence constraints, to be executed on a single-processor machine. Figure 1 shows worst-case computation time \( C_i \), absolute deadline \( d_i \), and precedence relations of each task \( \tau_i \) in \( \Gamma_2 \).

![Task Set](image)

Figure 1: Characteristics of the \( \Gamma_2 \) task set.

Show the minimum lateness schedule for \( \Gamma_2 \), using a Gantt chart.

Be sure to motivate your answer.

E3) Consider a task set \( \Gamma_3 \), composed of 5 periodic tasks \( \tau_1, \ldots, \tau_5 \) that share 4 resources a, b, c, d and execute on a single-processor machine. \( \Gamma_3 \)'s tasks are represented in Figure 2. Each resource is accessed in mutual exclusion using the Priority Ceiling Protocol (PCP).

![Resource Access](image)

Figure 2: Graphical representation of critical sections in \( \Gamma_3 \).

Figure 3 shows phase \( \Phi_i \), period \( T_i = D_i \), worst-case computation time \( C_i \), and a description of the access window to the shared resources of each task \( \tau_i \), in terms of start time \( t(R_k) \) and duration \( \delta_i R_k \) of the critical section for each task \( \tau_i \) and each resource \( R_k \).

![Resource Interactions](image)

Figure 3: Characteristics of the \( \Gamma_3 \) task set.

E3.1) What is the worst-case blocking time \( B_i \) for each task \( \tau_i \) in \( \Gamma_3 \)?

E3.2) Is \( \Gamma_3 \) feasible with RM+PCP?

E3.3) Using a Gantt chart, show the schedule under RM+PCP, from time 0 until completion of the first instance of \( \tau_5 \). Below the Gantt chart, show how \( \tau_5 \)'s active priority \( p_5 \) evolves.

Be sure to motivate your answer.