INTERNATIONAL UNIVERSITY - VIETNAM NATIONAL UNIVERSITY - HOCHIMINH CITY

MID-TERM EXAMINATION

Date: 2010-11-05

Duration: 80 minutes

Student ID: **Name:**

Subject: Operating Systems	
Dean of School of Computer Science and Engineering Signature: Dr. Nguyen Duc Cuong	**Lecturer** Signature: Dr. Tran Manh Ha

GENERAL INSTRUCTIONS

- This is open book examination
- Mobile phone & laptop are unallowed
- You write answers on this paper

GRADING SUMMARY

Problem	**Max. points**	**Act. points**
1	18	
2	30	
3	22	
4	18	
5	12	
Total	100	

Do not forget to enter your name and identification

Problem 1: *Processes and threads* (18 points)

Choose correct answers by marking the appropriate boxes.

true	false	
☐	☐	A system call is a request from a process to the kernel for a specific kernel service.
☐	☐	The address of the next instruction in the program to be executed is stored in the private user address space of the process.
☐	☐	A thread shares its kernel stack with other threads.
☐	☐	A thread can still keep a running state while its process is blocked.
☐	☐	A spinlock is a lock where a process or a thread waits in a loop ("spins") and repeatedly checks the lock available.
☐	☐	A race condition exists if the result produced by concurrent threads or processes depends unexpectedly on the order of their execution.

Solution:

true	false	
☒	☐	A system call is a request from a process to the kernel for a specific kernel service
☐	☒	The address of the next instruction in the program to be executed is stored in the private user address space of the process.
☐	☒	A thread shares its kernel stack with other threads.
☒	☐	A thread can still keep a running state while its process is blocked.
☒	☐	A spinlock is a lock where a process or a thread waits in a loop ("spins") and repeatedly checks the lock available.
☒	☐	A race condition exists if the result produced by concurrent threads or processes depends unexpectedly on the order of their execution.

Problem 2: *Mutual exclusion and synchronization* (30 points)

Suppose one phase in creating communication devices contains a concurrent processing of three processes: putter (P), checker (C) and getter (G). The P process puts chips on a device (using `put()` function), and places the device (using `append_B1()` function) on a shared chain B1; the C process takes the device (using `take_B1()` function) from the B1 chain, verifies (using `check()` function) and places (using `append_B2()` function) the device on a shared chain B2 if the device has no errors, drops if otherwise; the G process takes the device (using `take_B2()` function) from the B2 chain, and gets the device (using `get()` function) combined with another device.

Propose a solution for this problem using semaphores and the provided functions. The solution must satisfy requirements:

- Only the P process or the C process may access the B1 chain at any one time
- Only the C process or the G process may access the B2 chain at any one time
- The P, C and G processes take devices out of the chains one at time
- The P and C processes cannot add devices into the full chain
- The C and G processes cannot remove devices from the empty chain
- The chains are bounded buffers

```
/* program putter/checker/getter */
const int


semaphore




void putter() {
    while(true) {
        put();









    }
}

void getter() {
    while(true) {









        get();
    }
}
```

```
void checker() {
    while(true) {









    }
}
void main() {
    parbegin(                                );
}
```

Solution:

Applying the bounded buffer producer and consumer algorithm for the putter and checker processes, and for the checker and getter process along with a validation.

```
const int sizeofbuff1, sizeofbuff2
semaphore s1 = 1, n1 = 0, e1 = sizeofbuff1, // putter and checker
  s2 = 1, n2 = 0, e2 = sizeofbuff2; // checker and getter

void putter() {
    while(true) {
        put();
        semWait(e1);
        semWait(s1);
        append_B1();
        semSignal(s1);
        semSignal(n1);
}
}

void checker() {
    while(true) {
        semWait(n1);
        semWait(s1);
        taken_B1();
        semSignal(s1);
        semSignal(e1);

        if (check()) {
            semWait(e2);
            semWait(s2);
```

```
            append_B2();
            semSignal(s2);
            semSignal(n2);
        }
    }
}

void getter() {
    while(true) {
        semWait(n2);
        semWait(s2);
        taken_B2();
        semSignal(s2);
        semSignal(e2);
        get();
    }
}

void main() {
    parbegin(putter, checker, getter);
}
```

Problem 3: *Deadlock and starvation* (11+11 points)

A system uses the Banker's algorithm to avoid deadlock. Assume there are four processes and four resource type in the system. The maximum resource requirement and the current resource allocation matrices are

$$Claim = \begin{pmatrix} 4 & 4 & 2 & 1 \\ 4 & 3 & 1 & 1 \\ 13 & 5 & 2 & 7 \\ 6 & 1 & 1 & 1 \end{pmatrix} Allocation = \begin{pmatrix} 4 & 0 & 0 & 1 \\ 1 & 2 & 1 & 0 \\ 1 & 1 & 0 & 2 \\ 3 & 1 & 1 & 0 \end{pmatrix}$$

where $Claim_{ij}$ ($1 \leq i \leq 4$ and $1 \leq j \leq 4$) denotes the maximum requirement of process i for resource j; $Allocation_{ij}$ denotes the number of units of resource j that are currently allocated to process i. The total amount of each resource type in the system is given by the vector [16, 5, 2, 8].

1. Illustrate this state of the system (by filling in the empty tables with names)

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4

	R1	R2	R3	R4

(a) Initial state

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4

	R1	R2	R3	R4

(b) .....

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4

	R1	R2	R3	R4

(c) .....

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4

	R1	R2	R3	R4

(d) .....

Answer:

2. Illustrate whether the granting of a request by process 1 for 1 unit of resource 2 would be granted (by filling in the empty tables with names)

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4

	R1	R2	R3	R4

(a) Initial state

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4

	R1	R2	R3	R4

(b) .....

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4

	R1	R2	R3	R4

(c) .....

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4
P1				
P2				
P3				
P4				

	R1	R2	R3	R4

	R1	R2	R3	R4

(d) .....

Answer:

Solution:

1.

C	R1	R2	R3	R4
P1	4	4	2	1
P2	4	3	1	1
P3	13	5	2	7
P4	6	1	1	1

A	R1	R2	R3	R4
P1	4	0	0	1
P2	1	2	1	0
P3	1	1	0	2
P4	3	1	1	0

C-A	R1	R2	R3	R4
P1	0	4	2	0
P2	3	1	0	1
P3	12	4	2	5
P4	3	0	0	1

R All	R1	R2	R3	R4
	16	5	2	8

R Ava.	R1	R2	R3	R4
	7	1	0	5

(a) Initial state

C	R1	R2	R3	R4
P1	4	4	2	1
P2	0	0	0	0
P3	13	5	2	7
P4	6	1	1	1

A	R1	R2	R3	R4
P1	4	0	0	1
P2	0	0	0	0
P3	1	1	0	2
P4	3	1	1	0

C-A	R1	R2	R3	R4
P1	0	4	2	0
P2	0	0	0	0
P3	12	4	2	5
P4	3	0	0	1

R All	R1	R2	R3	R4
	16	5	2	8

R Ava.	R1	R2	R3	R4
	8	3	1	5

(b) P2 runs to completion

C	R1	R2	R3	R4
P1	4	4	2	1
P2	0	0	0	0
P3	13	5	2	7
P4	0	0	0	0

A	R1	R2	R3	R4
P1	4	0	0	1
P2	0	0	0	0
P3	1	1	0	2
P4	0	0	0	0

C-A	R1	R2	R3	R4
P1	0	4	2	0
P2	0	0	0	0
P3	12	4	2	5
P4	0	0	0	0

R All	R1	R2	R3	R4
	16	5	2	8

R Ava.	R1	R2	R3	R4
	11	4	2	5

(c) P4 runs to completion

C	R1	R2	R3	R4
P1	0	0	0	0
P2	0	0	0	0
P3	13	5	2	7
P4	0	0	0	0

A	R1	R2	R3	R4
P1	0	0	0	0
P2	0	0	0	0
P3	1	1	0	2
P4	0	0	0	0

C-A	R1	R2	R3	R4
P1	0	0	0	0
P2	0	0	0	0
P3	12	4	2	5
P4	0	0	0	0

R All	R1	R2	R3	R4
	16	5	2	8

R Ava.	R1	R2	R3	R4
	15	4	2	6

(d) P1 runs to completion

Answer: Safe

2.

C	R1	R2	R3	R4
P1	4	4	2	1
P2	4	3	1	1
P3	13	5	2	7
P4	6	1	1	1

A	R1	R2	R3	R4
P1	4	1	0	1
P2	1	2	1	0
P3	1	1	0	2
P4	3	1	1	0

C-A	R1	R2	R3	R4
P1	0	3	2	0
P2	3	1	0	1
P3	12	4	2	5
P4	3	0	0	1

R All	R1	R2	R3	R4
	16	5	2	8

R Ava.	R1	R2	R3	R4
	7	0	0	5

(a) Initial state

C	R1	R2	R3	R4
P1	4	4	2	1
P2	4	3	1	1
P3	13	5	2	7
P4	0	0	0	0

A	R1	R2	R3	R4
P1	4	1	0	1
P2	1	2	1	0
P3	1	1	0	2
P4	0	0	0	0

C-A	R1	R2	R3	R4
P1	0	3	2	0
P2	3	1	0	1
P3	12	4	2	5
P4	0	0	0	0

R All	R1	R2	R3	R4
	16	5	2	8

R Ava.	R1	R2	R3	R4
	10	1	1	5

(b) P4 runs to completion

C	R1	R2	R3	R4
P1	4	4	2	1
P2	0	0	0	0
P3	13	5	2	7
P4	0	0	0	0

A	R1	R2	R3	R4
P1	4	1	0	1
P2	0	0	0	0
P3	1	1	0	2
P4	0	0	0	0

C-A	R1	R2	R3	R4
P1	0	3	2	0
P2	0	0	0	0
P3	12	4	2	5
P4	0	0	0	0

R All	R1	R2	R3	R4
	16	5	2	8

R Ava.	R1	R2	R3	R4
	11	3	2	5

(c) P2 runs to completion

C	R1	R2	R3	R4
P1	0	0	0	0
P2	0	0	0	0
P3	13	5	2	7
P4	0	0	0	0

A	R1	R2	R3	R4
P1	0	0	0	0
P2	0	0	0	0
P3	1	1	0	2
P4	0	0	0	0

C-A	R1	R2	R3	R4
P1	0	0	0	0
P2	0	0	0	0
P3	12	4	2	5
P4	0	0	0	0

R All	R1	R2	R3	R4
	16	5	2	8

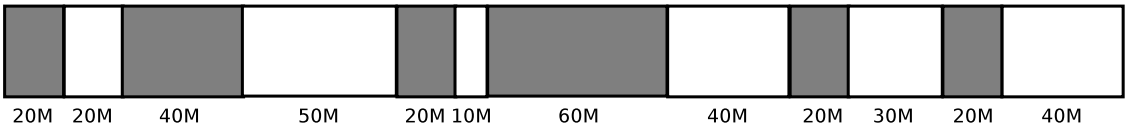
R Ava.	R1	R2	R3	R4
	15	4	2	6

(d) P1 runs to completion

Answer: Safe

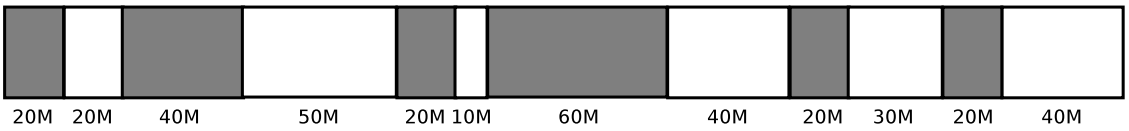
Problem 4: *Memory management* (6+6+6 points)

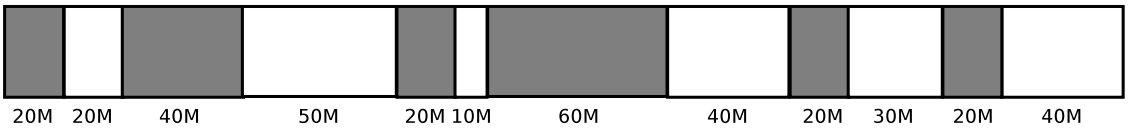
A dynamic partitioning scheme is being used, and the following is the memory configuration at a given point in time:

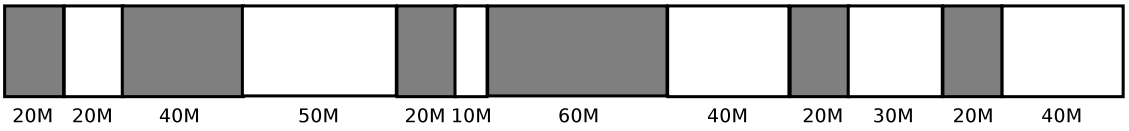


The shaded areas are allocated blocks; the white areas are free blocks. The next three memory requests are for 30M, 20M, and 10M. Indicate the starting address for each of the three blocks using the following placement algorithms:

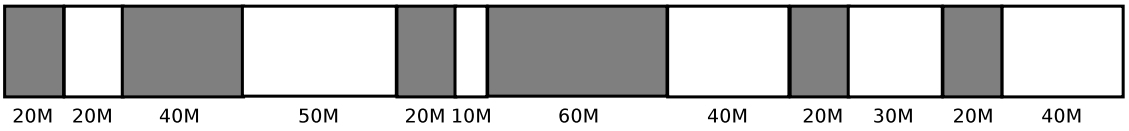
1. First-fit

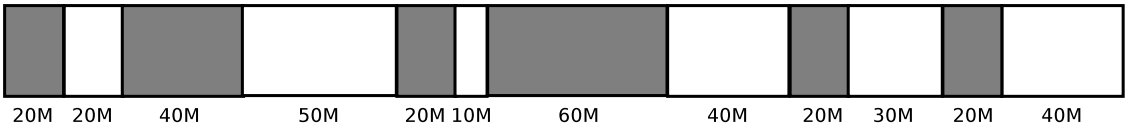


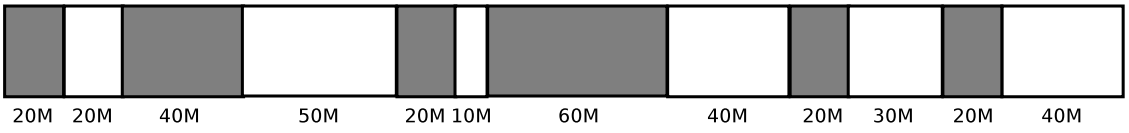




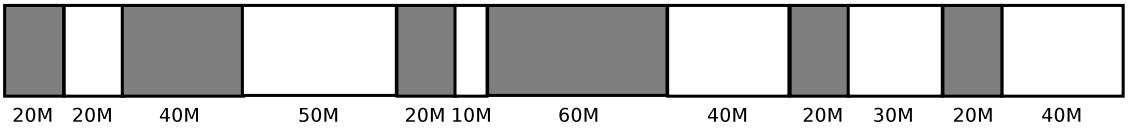
2. Best-fit

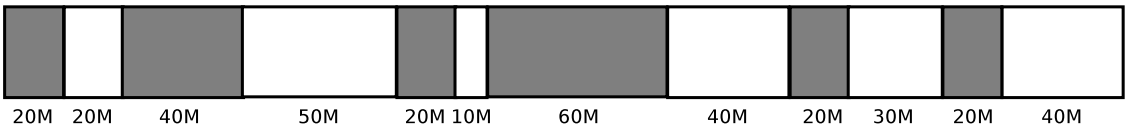


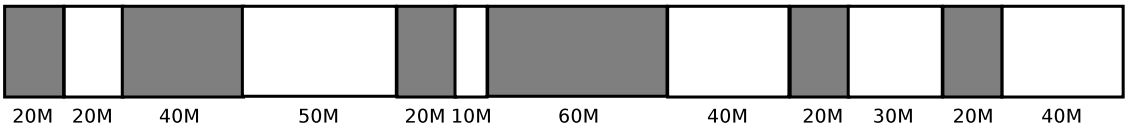




3. Next-fit. Assume the most recently added block is at the beginning of memory.

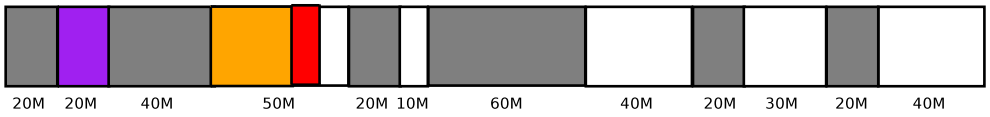


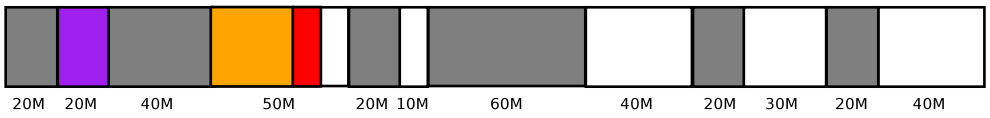


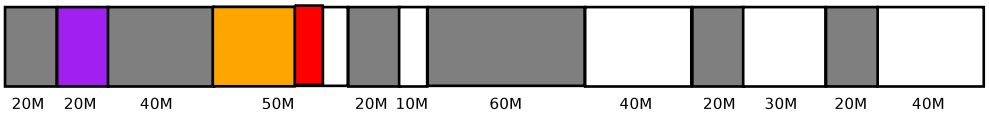


Solution:

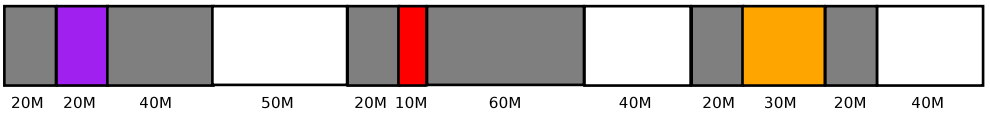
First fix

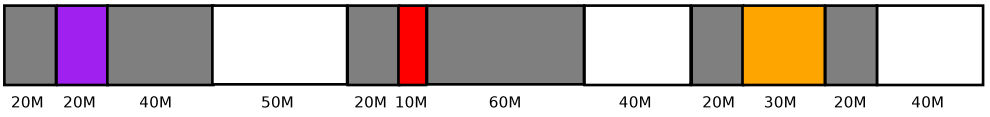


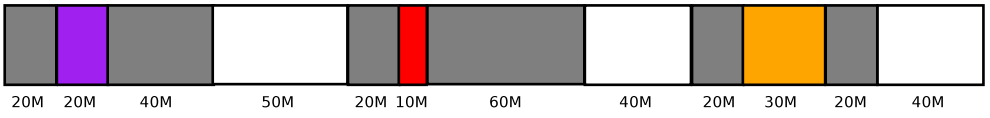




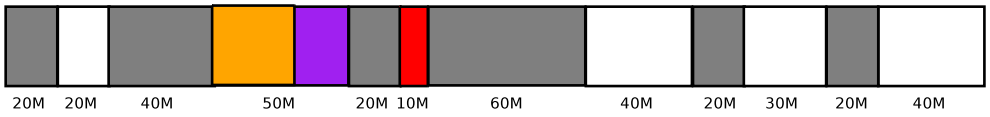
Best fix

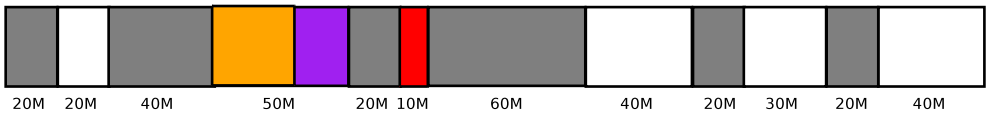


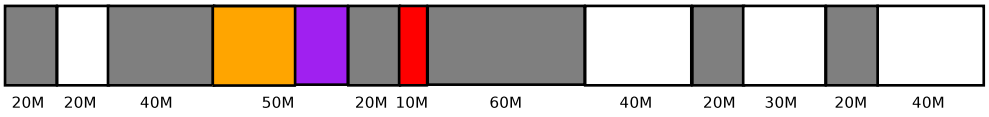




Next fit







Problem 5: *Process tree* (6+6 points)

Observe the C program shown below. Assume all system calls succeed (error handling code has been omitted for simplification).

```
#include <stdlib.h>
#include <unistd.h>

static void run() {
    fprintf(stdout, "a\n");
}

int main(int argc, char* argv[]) {
    fork();             /* fork() call #1*/
    run();
    if (fork() == 0) {  /* fork() call #2*/
        run();
    }
    return 0;
}
```

1. Identify the number of processes created by fork() calls and indicate the output of this program

2. Draw the process tree

Solution:

1. The fork() call #1 creates one new process. Two processes call run() to print out two character a. The two processes continue to invoke the fork() call #2 to create other two new children processes. Two children processes call run() to print out other two character a. In summary, there are four processes and the output is four characters a.

2. The fork() call #1 creates one new process, the fork() call #2 creates two new processes.