COMPE571 – Embedded Operating Systems – Fall 2019 – Homework 1

1 – Tasks and task states (6 pts)

In class, we learned that there are 3 major task states.

a) What are these task states?
b) List all feasible transitions between these states and explain why these transitions occur?
c) What are the possible extensions to these major task states?

2 – Deadlock (15 pts)

a) (5 pts) In class, we saw that there are two major ways to deal with deadlocks. List these and provide an example for each. Then, talk about the feasibility of their implementations in real systems.

b) (10 pts) Assume we have 4 processes, $\{P_1, P_2, P_3, P_4\}$, and 2 resources $\{R_1, R_2\}$. The current resource allocation matrix is follows:

Resources	**R_1**	**R_2**
P_1	1	3
P_2	1	2
P_3	1	2
P_4	2	0

How you should read this table: P_1 currently has 1 R_1 and 3 R_2 resources (similar for other processes). The following matrix shows the additional resource requests by each process:

Resources	**R_1**	**R_2**
P_1	1	2
P_2	4	3
P_3	1	7
P_4	5	1

How you should read this table: P_1, in addition to the resources it has, wants 1 R_1 and 2 R_2 resources (similar for other processes). Finally, the below is the availability vector of the resources:

$$\{R_1, R_2\} = \{1, 4\}$$

This vector shows how many of resource the system has in addition to the current resource allocations.
Question: Is the system with this current state deadlocked? What changes if we update the availability vector as $\{2, 4\}$.

This study source was downloaded by 100000896623665 from CourseHero.com on 02-18-2025 21:57:07 GMT -06:00

https://www.coursehero.com/file/55989351/hw11pdf/

3 – Processes (15 pts)

i) (5pts) How many times does the following program print ***hello***?

```
1.   #include <stdio.h>
2.   #include <unistd.h>
3.
4.   main() {
5.        int i;
6.        for (i = 0; i < 3; i++) {
7.             fork();
8.        }
9.        printf(“hello\n”);
10.       //DO SOME WORK
11.  }
```

ii) (10 pts) How can you modify the above code to differentiate all available processes in the system after line 10?

4 – Concurrency (10 pts)

For each program below, indicate whether or not it could have a deadlock, a race condition, or both. If so, explain why. Assume that the function **thread1** runs in one thread and the function **thread2** runs in another thread, and that the following data are shared between threads, and initialized as indicated prior to execution in each case:

```
semaphore m = 1;
semaphore n = 1;
   int x = 0;
   int y = 0;
```

a)

```
thread1() {
  wait(&m);
  x ++;
  wait(&n);
  y ++;
  signal(&n);
  signal(&m);
}
```

```
thread2() {
  wait(&n);
  x ++;
  wait(&m);
  y ++;
  signal(&m);
  signal(&n);
}
```

b)

```
thread1() {
  wait(&m);
  x ++;
  wait(&n);
  signal(&m);
  y ++;
  signal(&n);
}
```

```
thread2() {
  wait(&m);
  x ++;
  wait(&n);
  signal(&m);
  y ++;
  signal(&n);
}
```

This study source was downloaded by 100000896623665 from CourseHero.com on 02-18-2025 21:57:07 GMT -06:00

https://www.coursehero.com/file/55989351/hw11pdf/

5 – Semaphores (10 pts)

The semaphore wait and signal operations are defined as indivisible (or atomic) operations.

a) Why do they have to be indivisible?

b) Because they are indivisible does that mean that all other processes running on a multiprocessor system must stop when a wait or signal operation is executed? Explain why or why not.

6 – Process Synchronization (10 pts)

Show all possible output sequences resulting from running these two processes synchronously. Assume that the processes will not terminate before executing all the instructions. Explain your answer.

int x = 0; int y = 0;	"initialization"
Process A	**Process B**

while(x==0){};
printf("a");
y=1;
printf("d");
while(x==1){};
y=0;
printf("e");
y=1;
	printf("b");
	x=1;
	while(y==0){};
	x=0;
	while(y==0){};
	printf("c");

7 – Communicating Processes (10 pts)

Given the following communicating process network descriptions and the following input sequence:

(5 pts) i) explain how the network works by drawing a diagram and

(5 pts) ii) provide the output sequence.

Note: 1) Assume that the read operations are blocking (i.e. process cannot continue without getting the message from a buffer) and write operations are non-blocking. 2) You can assume that the channels have enough bandwidth so that there is no overwriting necessary.

```
Process p1 (input int a, output int x, output int y){
        int k;
        loop
                k = a.receive();
                if k mod 2 == 0 then
                        x.send(k);
                else
                        y.send(k);
                end if;
        end loop;
}
Process p2 (input int a, output int x){
        int k;
        loop
                k = a.receive();
                x.send(k);
        end loop;
}
```

This study source was downloaded by 100000896623665 from CourseHero.com on 02-18-2025 21:57:07 GMT -06:00

https://www.coursehero.com/file/55989351/hw11pdf/

```
Process p3 (input int a, input int b, output int x){
    int k;
    boolean alternate = false;
    loop
        if alternate then
            k = a.receive();
        else
            k = b.receive();
        end if;
        x = send(k);
        alternate = !alternate;
    end loop;
}

channel int I, O, C1, C2, C3, C4;
p1(I, C1, C2);
p2(C1, C3);
p2(C2, C4);
p3(C3, C4, O);
```

Input sequence = [3, 3, 6, 9, 11, 3, 6, 6, 1, 0, 3, 7]

8 - Synchronization with semaphores (12 pts)

The below shows the sequence of semaphore operations at the beginning and at the end of the tasks A, B, C.
Determine for the 4 cases, i, ii, iii, and iv given below, whether, and in which sequence, the tasks are executed, using the initializations of the semaphore variables given in the respective table. Each column under the cases corresponds to a different initialization with semaphore given values. For example, in case i), we have initially SA=2, SB=0, and SC=2. If a task gets stuck, point out where (in which line) it gets stuck.

Task A	**Task B**	**Task C**
wait(SA)	wait(SB)	wait(SC)
wait(SA)	…	wait(SC)
wait(SA)	…	wait(SC)
…	…	…
…	signal(SC)	…
…	signal(SA)	signal(SB)
signal(SB)		signal(SB)
END	**END**	**END**

Cases:

Semaphore	**Cases**			
	i)	**ii)**	**iii)**	**iv)**
SA	2	3	2	0
SB	0	0	1	0
SC	2	2	1	3

This study source was downloaded by 100000896623665 from CourseHero.com on 02-18-2025 21:57:07 GMT -06:00

https://www.coursehero.com/file/55989351/hw11pdf/

9 – Multitasking (12 pts)

Consider the C program below. (For space reasons, we are not checking error return codes, so assume that all functions return normally.)

```
main() {
      if (fork() == 0) {
            if (fork() == 0) {
                  printf("3");
            }
            else {
                  wait();
                  printf("4");
            }
      }
      else {
            if (fork() == 0) {
                  printf("1");
                  exit(0);
            }
            printf("2");
            wait();
      }
      printf("0");
}
```

Out of the 6 outputs listed below, choose only the valid outputs of this program. Assume that all processes run to normal completion. For the invalid outputs, explain why they are invalid. For the valid outputs, demonstrate an execution sequence that makes them possible. Selections with no explanations will not receive any credit.

a. 2030401 **b.** 1234000 **c.** 2300140 **d.** 2034012 **e.** 2130400 **f.** 4030120

This study source was downloaded by 100000896623665 from CourseHero.com on 02-18-2025 21:57:07 GMT -06:00

https://www.coursehero.com/file/55989351/hw11pdf/

No-credit Questions

1 – Monolithic kernel vs. microkernel

There are two major design types for operating system kernels: monolithic kernels and microkernels. Which of these types better satisfies the following requirements? If both are equally as good write both. Justify your answers.

a) Convenient access to operating system data structures
b) Addition of new operating system components
c) Modification of operating system components
d) Security and reliability

2 – Useful terms

Explain the following terms: mutual exclusion, deadlock, process vs. thread, priority inversion, and memory protection.

3 – Modern OS Requirements

In the early years of digital computers, programmers could work interactively on the machines. For many years following that time programmers had to submit their programs for processing at a later time and could no longer control the machines directly. What changes needed to be made to the hardware and operating systems that allowed programmers safe interactive access to the computers again on time-sharing systems? Also say why the changes were needed.

This study source was downloaded by 100000896623665 from CourseHero.com on 02-18-2025 21:57:07 GMT -06:00

https://www.coursehero.com/file/55989351/hw11pdf/