Q1	Q2	Q3	Q4	Q5	Tot

CEng 334 - Introduction to Operating Systems

Spring 2018-2019, Midterm I, Apr 4, 2019
(6 pages, 5 questions, 100 points, 100 minutes)

METU Honor Code and Pledge

METU Honor Code: The members of the METU community are reliable, responsible and Honorable people who embrace only the success and recognition they deserve, and act with integrity in their use, evaluation and presentation of facts , data and documents.

*I have read and understood the implications of the METU Honor Code. To be precise, I understand that this is a **closed** notes exam, and I am forbidden to access any other source of information other than provided within the exam. I will TURN OFF all my electronic equipment (phones, smart watches, etc.) and put it off the table along with other notes and materials that I may have with me. I understand that leaving electronic devices on during the exam is strictly forbidden. I understand and accept to obey all the rules announced by the course staff, and that failure to obey these will result in disciplinary action.*

Name: _____ **No:** _____ **Signature:** _____

QUESTION 1.(25 points)

a) (10 pts) After a context switch (between two processes of same program vs between two threads of the same process), Which of the following values will **change**?

Value	Process	Thread
Instruction pointer	✓	✓
Stack pointer	✓	✓
Process id	✓	
Virtual address of a global variable		
Pyhsical address of a global variable	✓	
Open files table	✓	

b) (15 pts) You need to implement `int startwithinput(program, filepath)` function that starts `program` in background that reads `filepath` as its stdin. Stdout of the program should be returned from the function so that caller can read it (yes. it is a pipe). Assume `program` parameter can directly be passed to `execl()` system call. In shell, it corressponds to "`program <file |` ". Where output of pipe is returned as a file handle.

```
int startwithinput(const char *prog[], const char *filepath) {
        int fd[2];
        pipe(fd);
        if (!fork()) {
                int f = open(filepath, O_RDONLY);
                dup2(fd[1],1);
                dup2(f, 0);
                close(fd[0]); close(fd[1]); close(f);
                execl(prog);
        } else {
                close(fd[1]);
                return fd[0];
        }
}
```

QUESTION 2.(25 points)

Assume you have a boat with capacity of two items. The load types are:
GRASS, SHEEP, WOLF, and MEAT.
We like to carry items to the other side of the river as a whole with the following rules:

- Sheep eat grass and wolves eat meat when they are loaded in the same boat. So we cannot put them together. However sheep-meat and wolf-grass match is acceptable. Two loads of the same type is also acceptable.
- Boat only starts the journey when two compatible items are loaded.
- All item threads call embark(LOADTYPE item) function to be loaded in a boat. embark() only returns when the boat is ready to go with the calling item in.
- An embarked item will call disembark(LOADTYPE) to leave the boat. When two items call disembark(), the boat will be empty and new items can be loaded (for simplicity, location of the boat is ignored)
- These functions/methods can be called repeatedly by mutliple item threads. Boat will keep carrying items with the rules above.
- Hint: Any embark request should block if there is no waiting compatible request. If there is one, it should release the waiting request and enter. When embarking successfull any new request should be rejected until two items disembarks.

Following functions/variables are made available for convenience:

```
typedef enum {GRASS,MEAT,WOLF,SHEEP} LOADTYPE;
int waiting[4] = { 0, 0, 0, 0};  // num. of waiting items of each type
LOADTYPE compatible(LOADTYPE current, int waiting[4]) {
        if (waiting[(current + 2) % 4] > 0) return (current+2) % 4;
        else if (waiting[current] > 0) return current;
        else return -1;
}
```

You are free to use semaphores or monitors in your solution (not both!). Implement only embark() and disembark() functions. You can use additional variables and definitions as you may need. Give a deadlock free solution that enforces the rules above.

This page will not be graded! Use next page for your answer.

```
sem mut = 1;
sem canenter[] = {0,0,0,0};
int inside = 0;
void embark(LOADTYPE current) {
        wait(mut);
    int other = compatible(current, waiting);
        if (other >= 0)  {
                signal(canenter[other]);
                inside += 2;
                waiting[other]--; // this one keeps holding mutex!!!
        } else {
                waiting[current]++;
                signal(mut);
                wait(canenter[current]); // directly enter
        }
}

void disembark(LOADTYPE current) {
        inside--;
        if (inside == 0)
                signal(mut);
}
```

Alternative Solution with Monitors

```
monitor Boat {
        int waiting[4] = {0,0,0,0};
        condition canenter[4], empty;
        int inside;
        LOADTYPE other;

        embark(LOADTYPE item) {
                while (inside > 0)
                        empty.wait();

                other = compatible(item);
                if (other >= 0) {
                        notify(canenter[other]);
                        inside++;
                } else {
                        waiting[item]++;
                        wait(canenter[item]);
                        waiting[item]--;
                        inside++;
                }
        }

        disembark(LOADTYPE item) {
                inside--;
                if (inside == 0) {
                        notifyAll(empty);
                }
        }
```

QUESTION 3.(15 points)

Consider a program which may print the following sequences:

- A1A2A3C1A4B1A5
- A1B1A2C1A3A4A5
- A1A2C1B1A3A4A5

but would never print the following sequences, with the violating portion underlined:

- B1A1A2A3A4A5C1
- C1B1A1A2A3A4A5
- A1C1B1A2A3A4A5
- A1B1A2A3A4C1A5
- A1B1C1A2A3A4A5

in its different executions. Note that each printf statement prints a letter followed by a digit, such as B1 or A5. Moreover, A1 is always printed before A2, and so on. Hint: The processes that print A, B and C sequences are different.

(5 points) Draw the process graph marking the vertices and edges of the graph with fork, printf, wait, and waitpid.

(10 points) Write the C program, using fork, wait and waitpid system calls, that would act as such.

```
int main() {
        int pidB, pidC;
        print("A1");
        if (pidB = fork()) {
                print("A2");
                if (pidC = fork()) {
                        printf("A3");
                        waitpid(pidC,0);
                        printf("A4");
                        waitpid(pidB,0);
                        printf("A5");
                } else
                        printf("C1");
        } else
                printf("B1");
}
```

Name:_____ **Id:**KEY!!_____

Name: _____ **No:** _____

QUESTION 4.(20 points)

Consider a system which has the following resources; 5 A's, 4 B's, 3C's represented as a triple: (5,4,3).

- We have three processes P1, P2 and P3 which have declared their maximum resource needs as (4,1,2), (2,3,2) and (4,1,2).
- Initially all processes have no resources.
- If a resource request is rejected, then the request is assumed to be canceled, and the process makes a different request later.
- When a process exits, it releases all of its resources.

How would the OS respond to the resource requests made by the processes in the following sequence using the Banker's algorithm? Write the decision of the OS and its reason behind it for each of the requests.

Process	Request	Result	Rejection reason or safe completion sequence	Remaining resource
Initial	—	—	—	(5,4,3)
P1	(1,1,1)	Granted	P1,P2,P3	(4,3,2)
P2	(2,1,1)	Granted	P1,P2,P3	(2,2,1)
P3	(1,1,1)	Rejected	no safe sequence with (1,1,0)	(2,2,1)
P1	(2,0,1)	Rejected	no safe sequence with (0,2,0)	(2,2,1)
P2	exits	—	—	(4,3,2)
P3	(3,0,1)	Granted	P3, P1	(1,3,1)
P1	(1,0,1)	Rejected	no safe sequence with (0,3,0)	(1,3,1)
P3	(1,1,0)	Granted	P3,P1	(0,2,1)
P1	exits	—	—	(1,3,2)
P3	exits	—	—	(5,4,3)

Name: _____ **No:** _____

QUESTION 5.(15 points)

(a) What does the POSIX standard describe?
POSIX is an open standard for Operating systems. It provides an API of system calls that are supported. POSIX compliant programs are portable among different POSIX-compliant OS's.The standard also describes the API of the Pthreads library.

(b) How does an OS handle exceptions, such as those caused by page faults, illegal instructions, I/O events? How is this similar to the implementation of system calls?
An OS getts the id of the exception and uses it to execute the corresponding handler in kernel mode. A process executes a exception generating instruction in the system which then causes the OS to jump to its implementation, and execute it by setting up a proper kernel context, and making a switch.

(c) What can a preemptive OS do, that a non-preemptive OS can't do?
A preemptive OS can forcefully take-away the CPU from a running process. Note: Forcefully taking-away a resource from a process is called preemption, but in the context of a "preemptive OS", we are talking only about CPU and not the other resources.

(d) In which aspects Starvation and Deadlock is similar, and in which aspects they are different? How can starvation happen in the Readers, Writers problem?
In both, some processes cannot make progress due to other processes using or holding the resources it's needing. In a deadlock situation, there are always more than one starving processes. However, starvation does not imply deadlock.
In Readers/Writers problem, if there is a high flow of incoming Readers, then the Writers may never be allowed to get in, and starve.

(e) What is the main difference between using a spinlocking and a mutex to wait for a resource?
In spinlocking, the waiting thread continues to use CPU in a while loop. In mutex, a waiting thread is put to sleep, until the resource is freed, and does not occupy the CPU.

(f) Process A that is executing on the processor, sends two consecutive SIGINT signals to Process B which is on the ready queue. Assuming that Process B has a signal handler installed for SIGINT which merely prints "Received SIGINT" on stdout. When and how does Process B react?
When: Process B reacts after the OS makes a context switch (not when the signal is sent). How: It reacts by printing "Received SIGINT" message ONCE, since signals are binary and are not queued.