Name: Student ID:

Q1	Q2	Q3	Q4	Total

İTÜ
Department of Computer Engineering
BLG 312E – Operating Systems 2021-2022 Spring Midterm Exam

1) (20p) Please briefly explain the following items:

a. Context switch

Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a context switch. When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.

b. Busy waiting

While a process is in its critical section, any other process that tries to enter its critical section must loop continuously. In fact, this type of mutex lock is also called a spinlock because the process "spins" while waiting for the lock to become available. This continual looping is clearly a problem in a real multiprogramming system, where a single CPU is shared among many processes. Busy waiting wastes CPU cycles that some other process might be able to use productively.

c. System calls

System calls provide an interface to the services made available by an operating system. These calls are generally available as routines written in C and C++, although certain low-level tasks (for example, tasks where hardware must be accessed directly) may have to be written using assembly-language instructions. The two main approach are to interact with operating system and to get operating system to perform a task for them.

d. Process Control Block (PCB)

Each process is represented in the operating system by a process control block (PCB)—also called a task control block. It contains many pieces of information associated with a specific process, including these: process state, program counter, CPU register, CPU-scheduling information, memory-management information, accounting information and I/O status information. In brief, the PCB simply serves as the repository for any information that may vary from process to process.

2) (25p) Explain the difference between Process and Thread by considering multi-process and multi-thread programming. Also, briefly state the pros and cons of kernel and user level threading.

A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals. A traditional (or heavyweight) process has a single thread of control. If a process has multiple threads of control which is called multi-thread, it can perform more than one task at a time. Multi-process uses two or more CPU if it needs, but multi-thread uses single CPU. In multi-process programming a separate address space for each process is created, whereas in multi-thread programming there is a common address space for all the threads.

User threads are supported above the kernel and are managed without kernel support, whereas kernel threads are supported and managed directly by the operating system.
Advantages of user level threading :
- Possible to have a separate scheduling algorithm for threads
- No space allocation required in the kernel for the thread table

- All calls are to local routines faster and has lower cost than making a call to a kernel routine (system call)

Disadvantages of user level threading :

- System calls causing the thread to be suspended cause all threads to be suspended
- Page fault
- Scheduling

Advantages of kernel level threading :

- Scheduling is handled by operating system.
- Page fault is not a problem : The kernel executes another ready thread of the same process.
- No need to modify system calls.

Disadvantages of kernel level threading :

- High cost of implementing and executing system calls.

İTÜ
Department of Computer Engineering
BLG 312E – Operating Systems 2021-2022 Spring Midterm Exam

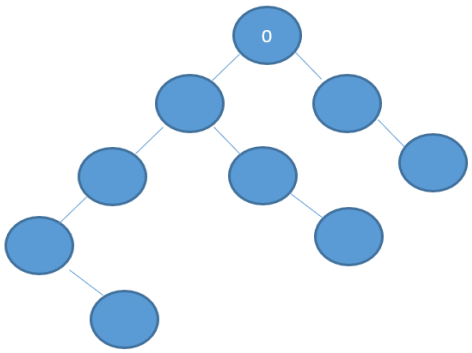
3) (25p) Please answer the following questions, for the given program below:

a. How many processes does the program create? Draw a tree diagram.

b. Give an example output.

```
static int x = 1;
int main() {
    int res;
    while(x <= 2) {
        res = fork();
        if(res == 0) {
            res = fork();
            printf("x=%d\n", x);
            x += 1;
        } else {
            x += 1;
        }
    }
}
```

(15p) This program creates 9 processes (1 parent, 8 child). Tree is an undirected graph with no cycles. Since it is a graph, each object must be shown exactly once. The resulting process hierarchy tree is given below:



(10p) This program may write 2 “x=1” and 6 “x=2” in any given order.

```
x=1
x=1
x=2
x=2
x=2
x=2
x=2
x=2
```

İTÜ
Department of Computer Engineering
BLG 312E – Operating Systems 2021-2022 Spring Midterm Exam

4) (30p)

a. List the three requirements, which any correct critical section solution should satisfy and briefly explain each of them.

Mutex: Only one process should be allowed to enter CS.
Progress: If CS is not in use, any waiting process should be allowed to use CS.
Bounded Waiting: No process should wait infinitely to access CS.

b. The following solution to the critical section problem for two processes is proposed by H. Hymann in 1966. Find a counterexample demonstrating that this solution is incorrect.

```
boolean blocked[2];
int turn;

void P(int id){
   while (TRUE) {
          blocked[id] = TRUE;
          while (turn != id) {
                 while ( blocked[1-id] );  /* wait - blocked*/
                 turn = id;
                 }
          /* Critical Section */
          blocked[id] = FALSE;
          /* Remainder Section */
   }
}

void main(){      /*Assign inital values and start two process P0 and P1*/
   blocked[0] = FALSE;
   blocked[1] = FALSE;
   turn = 0;
   parbegin(P(0),P(1));
}
```

Both enters CS (when P1 starts and switches to P0 before updating turn), MUTEX is not satisfied.