CS 219 Spring 2025 Quiz 1

(10 questions, 30 marks, 15% weightage)

Name:________________________ Roll number:________________________

Questions 1–10 carry 3 marks each. Write your answer neatly in the space provided.

1. What is the output of the following program? Write your answer next to the code.

```
int main() {
  int x = 5;
  int c1 = fork(); x--;
  int c2 = fork(); x--;
  if(c1 > 0) x--;
  if(c2 > 0) x--;
  if(c2 == 0 && c1 == 0)
    printf("grandchild x = %d\n", x);
  else if(c1 == 0)
    printf("child1 x = %d\n", x);
  else if(c2 == 0)
    printf("child2 x = %d\n", x);
  else
    printf("parent x = %d\n", x);
  wait(NULL); wait(NULL);
}
```

Ans: The following statements will be printed in any order.

```
parent x = 1
child1 x = 2
child2 x = 2
grandchild x = 3
```

2. What is the output of the following program? Write your answer next to the code.

```
void signal_handler() { printf("Signal received\n"); exit(0); }

int main() {
  int N = 3; int i = 0;
  while(i<N) {
    int cpid = fork(); i++;
    if(cpid == 0) {
      printf("child i=%d\n", i);
      signal(SIGINT, signal_handler);
      while(1);
    }
    else {
      sleep(1); kill(cpid, SIGINT); wait(NULL);
    }
  }
}
```

Ans: The following statements will be printed in the exact order. We will have partial marks (1 or 2 marks) for minor errors in the output (e.g., not incrementing "i").

```
child i=1
Signal received
child i=2
Signal received
child i=3
Signal received
```

3. What is the output of the following program when it is executed with the command-line argument of 5? That is, we run `./a.out 5` on the terminal. Write your answer next to the code.

   Note that the executable of the below program is called "a.out". The function `atoi` converts the command-line argument string to an integer, e.g., the code starts to run with `N = 5`. The function `sprintf` is used to print the new value of N for the child into a string, so that it can be given as the command-line argument to the next invocation of the program.

```
int sum = 0; int N = 0; //global variables
int main(int argc, char *argv[]) {
  N = atoi(argv[1]);
  sum += N; printf("sum=%d\n", sum);
  if(N < 5) return 0;
  for(int i=1; i<N; i++) {
    int ret = fork();
    if(ret == 0) {
      int child_N = N-i;
      char buf[20]; sprintf(buf, "%d", child_N);
      execlp("./a.out", "./a.out", buf, NULL);
    }
    else wait(NULL);
  } }
```

Ans: The following statements are printed.

```
sum=5
sum=4
```

```
sum=3
sum=2
sum=1
```

4. Process P1 forks P2 and P3. Process P2 forks P4 and P5. You are given that P4 has terminated (but not yet reaped), and all of the remaining processes are running. The processes above do not have any other children beyond those mentioned in the question. For each of the wait system call invocations below, specify if the wait system call blocks or returns. If it returns, describe the return value.

   (a) P2 calls wait (the default variant)
   Ans: returns with PID of P4

   (b) P1 calls wait (the default variant)
   Ans: blocks

   (c) P3 calls waitpid, where PID set to the PID of P4
   Ans: returns with an error code because P4 is not the child of P3

5. Consider a process P in xv6 that has just been context switched out after being forced to yield the CPU, following a timer interrupt. Draw a figure showing the kernel stack of the process right after the context switch. Clearly label the top of the stack, and all the important structures / contexts that are saved on the kernel stack.

   Ans: We expect the students to draw the trapframe (user context) at the bottom of the kernel stack, and the struct context (kernel context) at the top of the stack.

6. Consider a user C program that uses the `printf` function to print a line of text to the screen. The userspace C library invokes the `write` system call in the OS in order to perform this print task. The OS is POSIX compliant.

   (a) In which piece of software is the trap instruction invoked in this example? Your answer can be one of: user program, C library, OS `write` system call code, or OS code to set IDT.
   Ans: C library

   (b) Suppose we migrate this C program to another machine that also runs a POSIX-compliant OS and uses the same C library, but has a different underlying CPU architecture. Should the user code be rewritten, or recompiled, or can it remain unchanged, in order to work correctly?
   Ans: Should be recompiled to conform to the new ISA

   (c) Suppose we migrate this C program to another machine that also runs a POSIX-compliant OS on the same CPU architecture as part (a). However, a non-standard C library is being used, in which the format of arguments given to the printf function is different. Should the user code be rewritten, or recompiled, or can remain unchanged, in order to work correctly?
   Ans: Should be rewritten to follow the new printf function definition

7. Consider a process P1 that makes a blocking network read system call, and is context switched out. The OS switches to run process P2. While P2 is running (and still has some execution remaining on the CPU), network data arrives that can unblock P1, and the network device interrupts the OS. Assume the system runs on a single core CPU.

(a) What are the states of P1 and P2 just when the interrupt arrives?

Ans: P1 is blocked/sleeping and P2 is running

(b) If the OS were to be using a non-preemptive scheduler, would the OS handle the interrupt from the network card while P2 is still running? Answer yes/no with a brief justification.

Ans: Yes, because OS always handles the trap. A non-preemptive scheduler will not context switch out from P2, but will move to kernel mode in P2 to handle traps.

(c) If the OS were to be using a preemptive scheduler, is it guaranteed to always perform a context switch from P2 to P1, as soon as the network interrupt is handled and P1 becomes ready to run? Answer yes/no with a brief justification.

Ans: No, as some scheduling policies may decide to continue with P2 itself, e.g., a round robin policy where the time slice of P2 has not completed.

8. Consider three processes P1, P2, and P3 is a simple single core system. The arrival times of the processes are $t = 0$, $t = 2$, and $t = 3$ respectively. The CPU burst of P1 is 8 time units, and that of P3 is 4 time units. Let the CPU burst of P2 be X time units, for some positive integer X. The simple OS scheduler uses a preemptive Shortest Remaining Time First (SRTF) scheduling policy, with ties broken in favor of the process that is currently running on the CPU.

   (a) If $X = 3$, then at what time does process P3 complete?

   (b) For what values of X does process P2 finish before P1 and P3?

   (c) For what values of X does process P2 finish after P1 and P3?

   Ans: (a) $t = 9$ (b) $X \leq 5$ (c) $X \geq 6$

9. Consider the following events that occur when a parent process P makes a fork system call to spawn a child process C in an xv6 system. The OS goes back to the user mode of P after handling the system call. List the events in chronological order, from earliest to latest. For example, if you think event (a) occurs before event (b), and event (b) occurs before (c), then your answer should be "a, b, c".

   (a) Userspace context of P corresponding to general purpose CPU registers is saved on the kernel stack of P

   (b) Process C is marked as runnable/ready

   (c) Userspace context of P corresponding to EIP and ESP registers is pushed on to the kernel stack of P

   (d) The trap instruction is invoked on the CPU

   (e) The trap frame is popped from the kernel stack of P, and P returns to user mode

   (f) A new PCB is allocated for C, and a new memory image is created for C by copying from the memory image of P

   Write your answer here:

   Ans: d, c, a, f, b, e

10. Let us continue the previous question. Process P runs for some more time after the fork system call, when the timer interrupt goes off. P yields the CPU and is context switched out. The OS scheduler now switches to child process C. Arrange the following events in chronological order, from earliest to latest.

    (a) C pops the address of trapret from its kernel stack
    (b) P pushes userspace context (trapframe) onto its kernel stack
    (c) C pops the userspace context in the trapframe, and returns to user mode
    (d) P pushes kernel context onto its kernel stack
    (e) C pops kernel context from its kernel stack, and executes the code of forkret
    (f) The trap instruction is executed on the CPU

    Write your answer here:

    Ans: f, b, d, e, a, c