CS471: Operating Systems
Homework #1 - Due: February 8, 2026
Spring 2026
Points 30
Submit your work as a single pdf in Canvas.

Question 1 [Points 2] Including the initial parent process, how many processes are created by the program shown below? *Justify your answer*.

```c
int main()
{
    /* fork a child process */
    fork();
    fork();
    fork();
    int i;
    for (i = 0; i < 6; i++) fork();
    /* fork another child process */
    fork();
    return 0;
}
```

Question 2 [Points 4] Write a program using the fork() system call to do the following. The parent process (main program) forks a process (CHILD 1) to compute and print the sum of the n natural numbers (1,2,3,…,n) where n is a variable shared between the parent and CHILD 1. It also forks another process (CHILD 2) that finds the sum of cubes (^3) of the same first n natural numbers (1,2,3,…,n) where n is a variable it shares with the parent. Let CHILD 1 print "The sum of the first ** natural numbers is: ****". Let CHILD 2 print "The sum of the cubes of the first ** natural numbers is: ****". Have the parent invoke the wait () call to wait for both the child processes to complete before exiting the program. *Run and submit your code and output screenshot of the program for n=3 and n=10 for the following input.*

Question 3 [Points 3] Modify and run the program shown below in the following way. There is an array of 20 elements defined in the program. The elements of the array are: [20 18 16 14 12 10 8 6 4 2 -20 -18 -16 -14 -12 -10 -8 -6 -4 -2]. Thread 1 adds the first four elements [20, 18, 16, 14], Thread 2 adds the next four elements [12,10, 8, 6], …, Thread 5 adds the last four elements [-8, -6, -4, -2]. Finally, the sum of all the 20 elements is printed by the main program. The parent must wait for all threads to complete before printing the output. *Run and show the output.*

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS  3
int counter = 1;
void* PrintHello(void* threadid)
{
    counter = 2 * counter + (int)threadid;
    printf("\n Thread Id: %d  Counter:  %d\n", threadid, counter);
    pthread_exit(NULL);
}

int main(int argc, char* argv[])
{
    pthread_t threads[NUM_THREADS]; int rc, t;
    for (t = 0; t < NUM_THREADS; t++)
    {
        printf("Creating thread %d\n", t);
        rc = pthread_create(&threads[t], NULL, PrintHello, (void*)t);
        if (rc)
        {
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(1);
        }
    }
    printf("\n Counter:  %d\n", counter);
    pthread_exit(NULL);
}
```

Question 4 [3 points] Run the following program and identify the pid (Process ID) of the parent, child, and grandchild processes. Briefly explain what the program is doing. *Run and show output.*

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
    pid_t pid, pid1, pid2, pid0, pid3, pid4, pid5;
    /* fork a child process */
    pid = fork();
    if (pid < 0)
    {
        /* error occurred */
        fprintf(stderr, "First Fork Failed");
        return 1;
    }
    else if (pid == 0)
    {
        /* child process */
        pid1 = getppid();
        pid2 = getpid();
        printf("Child PID = %d\n", pid2);
        printf("Parent PID = %d\n", pid1);
        pid3 = fork();
        if (pid3 < 0)
        {
            fprintf(stderr, “Second fork failed”);
            return 1;
        }
        else if (pid3 == 0)
        /*Grandchild process */
        {
            pid4 = getpid();
            pid5 = getppid();
            printf("Grandchild PID = %d\n", pid4);
            printf("Child PID = % d\n", pid5);
        }
    }
    else
    {
        /* parent process */
```

```
        pid1 = getpid();
        pid0 = getppid();
        printf("Parent: = %d\n", pid1);
        printf("Root: = % d\n", pid0);
        wait(NULL);
    }
    return 0;
}
```

Question 5 [Points 3] Compare user-level threads and kernel-level threads. What are the advantages and disadvantages of each?

Question 6 [Points 3] Consider the following set of six processes, with the CPU burst time (in milliseconds).

Process	Burst time
P1	30
P2	5
P3	22
P4	10
P5	25
P6	50

Draw two Gantt charts that illustrate the execution of these processes using the ***FCFS and SJF*** scheduling algorithms. Compute the average wait time for each.

Question 7 [Points 2] Consider the following set of six processes, with the CPU burst time (in milliseconds), and priority.

Process	Burst time	Priority
P1	30	1
P2	5	2
P3	22	3
P4	10	1
P5	25	2
P6	50	3

Draw a Gantt charts that illustrate the execution of these processes using the ***priority*** scheduling algorithms (a smaller priority number implies a higher priority) and compute the average wait time.

Question 8 [Points 2] Consider the following set of six processes, with the CPU burst time (in milliseconds).

Process	Burst time
P1	30
P2	5
P3	22
P4	10
P5	25
P6	50

Draw a Gantt charts that illustrate the execution of these processes using the ***RR*** (quantum=20) scheduling algorithms and compute the average waiting time.

Question 9 [Points 4] Write a *program* that takes as input a list of jobs (processes with their arrival time and CPU burst time and schedules them based on FCFS policy. It outputs the <Process Id, arrival time, CPU burst time, completion time> for each of the jobs. Assume that the jobs are listed in the order of their arrival time. *Run and submit your code and output screenshot for the following input.*

Input:

ProcessID	Arrival	CPU Burst
1	10	20
2	35	5
3	45	10
4	50	50

For the above sample input, the output would be:

ProcessID	Arrival	CPU Burst	Completion
1	10	20	30
2	35	5	40
3	45	10	55
4	50	50	105

Question 10 [Points 4] Write a multithreaded program that outputs prime numbers. This program should work as follows: The user will run the program and will enter a number on the command line. The program will then create a separate thread that outputs all the prime numbers less than or equal to the number entered by the user. *Run and submit your code and output screenshot of the program.*