CS342 Operating Systems
Spring 2024
Homework #2

Assigned: Mar 9, 2024.
Due date: Mar 17, 2024. Document version: 1.0

Q1. We have the following program. What are the pairs of values printed out by this program?

```
int a = 300;
int b = 100;
int ret;
int main()
{
        ret = fork();
        a = a + 10;
        if (ret == 0) {
                a = a * 2;
                b = b + 50;
                printf ("%d %d\n", a, b);
                if (fork() == 0) {
                    a = a + 1000;
                    printf ("%d %d\n", a, b);
                    exit(0);
                }
                b = b + 10;

        }
        else {
                a = a + 200;
                b = 7 * b;
                printf ("%d %d\n", a, b);
                b = b + 80;
        }
        printf ("%d %d\n", a, b);
}
```

Q2. Consider the following program. What are the values that are printed out by this program?

```
unsigned long x = 1000;
unsigned long y = 2000;
unsigned long z = 400;

void * func1(void *p)
{
    unsigned long x, z;

    x = 1300;
    y = 3000;
    z = *((unsigned long *)p);
    z = z +  300;
    *((unsigned long *)p) += 500;
    printf ("%lu %lu %lu\n", x, y, z);
    pthread_exit(NULL);
}

int
main()
{
    pthread_t tid;
    unsigned long a = 5000;


    pthread_create (&tid, NULL, &func1, (void *)&a);
    pthread_join(tid, NULL);

    printf ("%lu %lu %lu\n", x, y, z);
}
```

Q3. Consider the following program. Find out the triples printed out by this program. What can you say about their order?

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

unsigned long a=10;
unsigned long b=20;
#define COUNT 5

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void * func(void *p)
{
    unsigned long a = 50;

    pthread_mutex_lock(&lock);
    a = a + 1;
    b = b + 1;
    pthread_mutex_unlock(&lock);

    printf ("%lu %lu %lu\n", a, b, (unsigned long)p);
    pthread_exit(NULL);
}


int main()
{
    pthread_t tids[COUNT];
    unsigned long k;
    int ret;


    for (k = 0; k < COUNT; ++k)
        pthread_create (&(tids[k]), NULL, &func, (void *) (k+1));
    for (k = 0; k < COUNT; ++k)
        pthread_join(tids[k], NULL);
    printf ("%lu %lu %lu\n", a, b, k);

    exit (0);
}
```

Q4. Schedule the following processes with RR (q=10), SJF, FCFS, and SRTF. For each process, find out the finish (completion) time, turnaround time, waiting time, response time (which is defined as the time elapsed until the first execution). Compare the algorithms with respect to average turnaround time.

	Arrival time	CPU time (burst length)
A	0	100
B	15	80
C	25	100
D	45	60
E	65	45

Q5. Consider an MLFQ scheduling algorithm with the following rules (different MLFQ algorithms may have different rules; the rules here are just for this example). There are 3 queues corresponding to 3 priority levels: H (high priority), M (medium priority), and L (low priority). Assume all queues are using round robin scheduling. The time slice (time quantum) is 1 time unit for for H queue, 2 time units for M queue, and 4 time units for L queue. Assume the following scheduling rules.

- If a running process X is preempted by a higher priority process, X is put to the tail of its queue at the same level. Next time it is again given a whole time slice.
- If a process X uses its entire time slice in a queue (i.e., cpu burst did not finish when time slice expired, or burst finished exactly when time slice expired), its priority (queue) is reduced by one level. If the process X is already in L queue, it stays in L queue.
- If a process goes for I/O before using its entire time slice, it is kept in the same queue. That means after returning from I/O, it is added back to the same queue (to the tail).
- If there is a tie, i.e., two processes arrive to a queue at the same time, IDs of the processes are considered (for example, A comes before B).
- Every 25 time units (after 25 units passed), priorities of all processes are boosted to H queue immediately. The currently running process is preempted and the next one in the H queue is run. Next time, the preempted process is given a whole time quantum.

Now consider the following 3 processes: A, B, and C. All processes arrive at time 0 and are placed to the H queue in the following order: A, B, C (A is head). Process A wants to run (in cpu) for 2 time units, then wants to do I/O for 5 time units (waiting 5 time units), and wants to repeat this behavior in total 4 times. (Process B wants to run 3 times units, then wants to do I/O for 4 time units, and wants to repeat this behavior (in total 3 times). Process C has a single burst of length 15 time units.

Schedule these processes with MLFQ algorithm. Show the schedule in a chart or table.

Q6. Consider a CPU that can serve (i.e., execute) 28 bursts per second on the average ($\mu = 28$). Assume burst times are exponentially distributed. The burst inter-arrival times are also exponentially distributed with mean 40 ms.

a) What is the CPU utilization?
b) What is the probability that the CPU is idle?
c) What is the average response time (E[R])?
d) What is the average number of bursts in the ready queue (not including the one being executed in the CPU)?
e) What is the average waiting time in the queue (not considering the execution time)?
f) Assume we want to decrease the average response time by 50 percent (that means to half). How much should we decrease the arrival rate (i.e., the throughput) to cause this decrease in the response time?

Q7. Write two disadvantages and one advantage of the user-space implementation of threading support compared to kernel-space implementation.

Q8. Assume we have a file F that can be accessed by multiple threads if the sum of the IDs of the threads is less than or equal to a fixed number N. Each thread has a unique id. Synchronize the threads with mutex and condition variables. That means implement two functions: request() and leave(). A thread will use them as follows:

```
request() // may block the caller
// access file F
release()
```

Your solution does not have to be starvation-free.

Q9. Assume we have binary semaphores supported by an OS (applications can use them), but not general semaphores. Think about and try to show (in pseudo-code) how general semaphores can be implemented (at application level) with binary semaphores (given by OS).