CS 3733 Operating Systems: Assignment 5 - new -HELP

MainThread

```
process command line args and get the simulation parameters (e.g., ALG, QUANTUM, InputFile)
create/initialize the necessary data structures (Ready_Q and IO_Q (double linked lists of PCB),
        file_read_done=0, cpu_sch_done = 0,  io_sys_done = 0, cpu_busy=0, io_busy=0, sem_cpu=0, sem_io=0)
create start the following three threads with appropriate parameters
wait until all threads are done
print performance metrics
```

```
struct PCB {
  int PID, PR;
  int  numCPUBurst, numIOBurst;
  int *CPUBurst,   *IOBurst;  /* to create
        dynamic arrays to store cpu and io burst times */
  int  cpuindex,    ioindex;
  struct timespec ts_begin, ts_end;
  struct PCB *prev, *next;
 // more fields for performance measures
// use the system time to determine how much waited etc.
}
```

FileRead thread

```
get the file name, open it; currPID=0;
while( not EOF)
    read a line
    if proc,   create a PCB structure withPID=++currPID,
               read other parameters into it,
               insert PCB into Ready_Q
               sem_post(&sem_cpu)
    if sleep, simply let this thread usleep for the given ms
    if stop, break
file_read_done = 1
```

```
clock_gettime(
CLOCK_MONOTONIC,
&PCB->ts_begin);
```

CPU scheduler thread

```
while(1)   //  This is for FIFO. Similarly, You need to develop other algorithms SJF, RR, PR
    if Ready_Q is empty && !cpu_busy && IO_Q is empty && !io_busy && file_read_done is 1, then break!
    if (ALG is FIFO)
        res = sem_timedwait(&sem_cpu, &atimespec /* say 1 sec */ );
        if(res==-1 && errno==ETIMEDOUT) continue;
        cpu_busy = 1
        get (remove) the first PCB from Ready_Q
        usleep for PCB->CPUBurst[PCB->cpuindex]  (ms)
        PCB->cpuindex++
        if PCB->cpuindex >= PCB->numCPUBurst // this is the last cpu burst
            terminate this PCB;  cpu_busy = 0
        else
            insert PCB into IO_Q
            cpu_busy = 0
            sem_post(&sem_io)
cpu_sch_done = 1
```

```
clock_gettime(CLOCK_MONOTONIC,&BPC->ts_end);
elapsed =  PCB->ts_end.tv_sec -
           PCB->ts_begin.tv_sec;
elapsed += (PCB->ts_end.tv_nsec -
       PCB->ts_begin.tv_nsec) / 1000000000.0;
printf("turnaround = %f ms\n", elapsed*1000);
```

I/O system thread

```
while(1)  //  this is always FIFO
    if Ready_Q is empty && !cpu_busy && IO_Q is empty && file_read_done is 1, then break!
    res = sem_timedwait(&sem_io, &atimespec /* say 1 sec */ );
    if(res==-1 && errno==ETIMEDOUT) continue;
    io_busy = 1;
    get (remove) the first PCB from IO_Q
    usleep for PCB->IOBurst[PCB->ioindex]     (ms)
    PCB->ioindex++
    insert PCB into Ready_Q
    io_busy = 0
    sem_post(&sem_cpu)
io_sys_done = 1
```

You need to figure out how to synchronize/coordinate these threads, protect critical sections, how to collect data to report performance metrics, and other implementation details error/exception handlings....

Note1: Some students asked about a possible case of deadlock when waiting on Ready_Q or IO_Q.
An easy way to deal with that would be to use sem_timedwait() rather than sem_wait(…); which, I have included as a possible solution along with some other coordination mechanisms in the above high-level solution.

Note2: To implement sleep for some ms, you can use **usleep();** please see its man page and the sample program below.

Note3: I did not do anything about critical sections in the above high-level solution, you need to identify and protect them! Also, you need to figure out how to collect data!

Note4: When collecting data, simply use the system time to measure delays, round around, and waiting in ready queue times etc. For example, when a PCB is put into Ready_Q save the system time (e.g., PCB->timeEnterReadyQ = getsystemtime();) then later when CPU gets that PCB from Ready_Q, we can simply determine waiting time by wait_time = getsystemtime() - PCB->timeEnterReadyQ; Of course, you need to accumulate all the wait_times for a process to find its total waiting time! Also at the end, we need to keep track of all total waiting times to the average waiting time! I used getsystemtime(); as a generic name here is a sample program showing how you can get system time!

```
// s2.c  test program to illustrate how to get system time.
// gcc s2.c -o s2
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/times.h>

int main(int argc, char *argv[]){

     struct timespec ts_begin, ts_end;
     double elapsed;
     long sleep_time_ms;

     if (argc < 2) {
          printf("Usage: %s sleep_time_ms \n", argv[0]);  return 0;
     }

     sleep_time_ms = atoi(argv[1]);

     printf("sleep %ld ms...\n", sleep_time_ms);

     clock_gettime(CLOCK_MONOTONIC, &ts_begin);    // getsystemtime();

     usleep(sleep_time_ms*1000);

     clock_gettime(CLOCK_MONOTONIC, &ts_end);      // getsystemtime();

     elapsed = ts_end.tv_sec - ts_begin.tv_sec;
     elapsed += (ts_end.tv_nsec - ts_begin.tv_nsec) / 1000000000.0;

     printf("elepsed time =  %.3lf ms\n\n", elapsed*1000);

     return 0;

}
```