SJSU CS 149 HW4 SPRING 2021

REMINDER: Each homework is individual. "Every single byte must come from you." Cut&paste from others is not allowed. Keep your answer and source code to yourself only - never post or share them to any site in any way.

[Type your answer. Hand-written answer is not acceptable.]

1. (20 pts) Consider the following set of processes, with the length of the CPU burst time given in milliseconds:

Process	Burst Time	Priority
*P*1	2	3
*P*2	1	1
*P*3	8	5
*P*4	4	2
*P*5	5	4

The processes are assumed to have arrived in the order *P*1, *P*2, *P*3, *P*4, *P*5, all at time 0.
a. Use any software to draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, nonpreemptive SJF, nonpreemptive priority (a larger priority number implies a higher priority), and RR (quantum = 2). Hand drawing is **not** accepted.
b. For each algorithm in a, calculate the waiting time for each process, and the average waiting time.
c. For each algorithm in a, calculate the turnaround time for each process, and the average turnaround time.
d. Specify these four Gantt charts in text (for online exam).

2. (80 pts) [programming question] SJSU Kindergarten takes care of little boys and girls for SJSU employees and students. The kindergarten allows boys and girls to play in a playroom with PlayStation 5.

The boys and girls
There are `NUM_OF_BOYS` boys and `NUM_OF_GIRLS` girls in the kindergarten. Boys and girls share a single classroom where they can eat. Each boy alternates between eating in the classroom and playing in the playroom (each must start with eating). So do girls. To prevent boys and girls from fighting with each other in the playroom, the kindergarten sets up the following rules:

- There cannot be boys and girls in the playroom at the same time.
- There should never be more than `MAX_ROOM_CAP` boys (or girls) in the playroom since there are only limited number of PlayStations.

Your solution must avoid deadlock. For now, do not worry about starvation.

Use Pthreads to create one thread per boy and one thread per girl. Each boy eats from 1 up to `MAX_EAT_TIME` seconds, and plays from 1 up to `MAX_PLAY_TIME` seconds. So does each girl. To simulate boy (or girl) eating or playing in threads, the appropriate threads should invoke `sleep()` for a random period of time. Instead of invoking the random number generator `rand()` which is not reentrant, each thread must invoke the reentrant version `rand_r()` before each invocation to `sleep()`.

- `rand_r()` computes a sequence of pseudo-random integers in the range [`0`, `RAND_MAX`]. If you want a value between `1` and `n` inclusive, specify `(rand_r(&seed) % n) + 1`.
- Use a different seed value for `rand_r()` in each thread so that each thread can get a different sequence of pseudo-random numbers.
- Each time before a thread invokes `sleep()`, the thread must invoke `rand_r()` first.

For simplicity, each boy or girl thread terminates after playing games `NUM_OF_GAMES` times. The main program invokes `pthread_join` to wait for the termination of all threads and then the entire program terminates.

POSIX Synchronization
Cut&paste the followings to your source code:

```
/* the maximum time (in seconds) to eat and play */
#define MAX_EAT_TIME  3
#define MAX_PLAY_TIME 2
/* number of boys, girls */
#define NUM_OF_BOYS   3
#define NUM_OF_GIRLS  4
/* # of games each boy or girl must play before terminate */
#define NUM_OF_GAMES  2
/* playroom cap */
#define MAX_ROOM_CAP  2

/* semaphores and mutex lock */
pthread_mutex_t  boys_mutex, girls_mutex;
/* the number of waiting boys, girls */
Int boys, girls;
/* counting semaphore - boys, girls waiting to enter playroom */
sem_t boys_q, girls_q;
/* binary semaphore - playroom */
sem_t room_mutex;
```

Other than the above global shared variables, you are **not** allowed to have any additional global variables.

The global variable `boys` and `girls` keep track of the number of boys and girls who wish to play games, respectively. They are protected by `boys_mutex` and `girls_mutex`, respectively. The binary semaphore `room_mutex` is locked if the room is occupied (by either boys or girls) and unlocked otherwise. Counting semaphores `boys_q` and `girls_q` are *unnamed semaphores* to ensure that there are no more than `MAX_ROOM_CAP` boys (or girls) in the playroom at a time.

After a boy eats for a while, a boy arrives at the door of the playroom. One should increase the value of `boys`. When the first boy arrives, he acquires the `room_mutex`, barring girls. Any boy (not just the first one) then waits for the counting semaphore `boys_q` to enter the playroom - a boy may enter immediately if there are less than `MAX_ROOM_CAP` boys in the room, or else waits in the queue (stand in line). After each boy in the playroom plays for a while, a boy leaves the playroom by signaling `boys_q`. One should then decrease the value of `boys`. When the last boy leaves the playroom (no boys standing in line), he releases the `room_mutex`, allowing girls (or boys) to enter. Girls do likewise by using corresponding variables.

Note the value of `boys` include the ones currently standing in line outside of the playroom as well as the ones already in the playroom. The only purpose of a such variable is to identify if this is the first boy to enter the playroom, or the last boy to leave the playroom. The same also applies to `girls`.

Your program output format must be similar to the followings (the exact sequence is different, of course):

```
kong@ubuntu2004:~/tmp$ ./playroom
CS149 Spring 2021 PlayRoom from FirstName LastName
boy[0, 0]: eat for 3 seconds
boy[1, 0]: eat for 2 seconds
boy[2, 0]: eat for 1 seconds
girl[0, 0]: eat for 1 seconds
girl[1, 0]: eat for 2 seconds
girl[2, 0]: eat for 3 seconds
girl[3, 0]: eat for 1 seconds
girl[3, 0]: arrive, girls (including me) = 1
girl[3, 1]: play for 1 seconds
boy[2, 0]: arrive, boys (including me) = 1
girl[0, 0]: arrive, girls (including me) = 2
girl[0, 1]: play for 2 seconds
girl[1, 0]: arrive, girls (including me) = 3
girl[3, 1]: depart, girls (excluding me) = 2
girl[3, 1]: eat for 2 seconds
girl[1, 1]: play for 1 seconds
girl[2, 0]: arrive, girls (including me) = 3
girl[0, 1]: depart, girls (excluding me) = 2
girl[0, 1]: eat for 3 seconds
girl[2, 1]: play for 1 seconds
girl[1, 1]: depart, girls (excluding me) = 1
girl[1, 1]: eat for 2 seconds
girl[3, 1]: arrive, girls (including me) = 2
girl[3, 2]: play for 1 seconds
girl[2, 1]: depart, girls (excluding me) = 1
girl[2, 1]: eat for 3 seconds
girl[3, 2]: depart, girls (excluding me) = 0
boy[2, 1]: play for 1 seconds
boy[1, 0]: arrive, boys (including me) = 2
boy[1, 1]: play for 2 seconds
boy[0, 0]: arrive, boys (including me) = 3
girl[1, 1]: arrive, girls (including me) = 1
boy[2, 1]: depart, boys (excluding me) = 2
boy[2, 1]: eat for 3 seconds
boy[0, 1]: play for 1 seconds
boy[1, 1]: depart, boys (excluding me) = 1
boy[1, 1]: eat for 2 seconds
boy[0, 1]: depart, boys (excluding me) = 0
boy[0, 1]: eat for 1 seconds
girl[1, 2]: play for 1 seconds
girl[0, 1]: arrive, girls (including me) = 2
girl[0, 2]: play for 1 seconds
girl[2, 1]: arrive, girls (including me) = 3
boy[0, 1]: arrive, boys (including me) = 1
girl[0, 2]: depart, girls (excluding me) = 2
girl[1, 2]: depart, girls (excluding me) = 1
girl[2, 2]: play for 1 seconds
girl[2, 2]: depart, girls (excluding me) = 0
boy[0, 2]: play for 2 seconds
boy[1, 1]: arrive, boys (including me) = 2
boy[1, 2]: play for 1 seconds
boy[2, 1]: arrive, boys (including me) = 3
boy[1, 2]: depart, boys (excluding me) = 2
boy[2, 2]: play for 2 seconds
boy[0, 2]: depart, boys (excluding me) = 1
boy[2, 2]: depart, boys (excluding me) = 0
main: done
kong@ubuntu2004:~/tmp$
```


- boy[a, b]: a is boy ID (0, 1, etc.), b is # of games played by boy a.
- girl[c, d]: c is girl ID (0, 1, etc.), d is # of games played by girl c.

- For a given a the value of b in boy[a, b] keeps increasing until reaching `NUM_OF_GAMES`.
- For a given c the value of b in girl[c, d] keeps increasing until reaching `NUM_OF_GAMES`.
- The # of boys (or girls) in the sample output is the exact value of the global variable `boys` (or `girls`) at that moment. "including me" and "excluding me" imply that we print out its value after modifying its value.
- "arrive" means the one only arrives at the gate of the playroom; it does NOT mean one already enters the playroom. In particular, the first "boy[2,0]: arrive" in the above sample output means boy #2 is outside the playroom and is waiting to enter the playroom. boy #2 cannot enter immediately because some girls are already in the playroom; boy #2 has to wait outside.
- "depart" means he (or she) finishes playing and leaves the playroom.

Reminders

1. Each invocation the program always prints out "CS149 Spring 2021 PlayRoom from FirstName LastName" only once. Take screenshots of the entire program execution (including "CS149 Spring 2021 PlayRoom from …"). It is OK to have several screenshots. The last screenshot must capture the end of program execution.

2. Any API in a multi-threaded application must be thread-safe and reentrant (e.g., call `rand_r()` instead of `rand()`). Invoking any non thread-safe or non reentrant API is subject to deduction.

3. You are not allowed to have any additional global variables other than those aforementioned global variables.

4. Before using any mutex, semaphore, and condition variable, one must initialize it. One must destroy any mutex, semaphore, and condition variable before the process terminates.

5. You are not allowed to call `sem_getvalue()` for any semaphore.

6. One does not need to implement an explicit waiting queue. The default Pthread implementation on Linux wakes up threads waiting in a semaphore or condition variable in FIFO order. To avoid I/O buffer flushing bug, add "`fflush(NULL);`" after each `printf` in the program.

7. One must invoke the random number generator each time to get any eating time, and playing time, of boys and girls (thread calls `sleep()`). Other than playing in the game room (simulated by calling `sleep()`), do not call `sleep()` within any critical section.

8. You must utilize array for various data structure and must utilize loop to remove repeating code. Any repetitive variable declaration and/or code are subject to deduction.

9. No recursion is allowed in any function.

10. Best practice:

a. include necessary header files only.

b. remove any warning messages in compilation.

c. error handling: always checks the return code from any API.

Compile your program with "`gcc -o playroom playroom.c -pthread`". You can execute the program in a Linux terminal with "`./playroom`".

What to do

a. (15 pts) screenshots of stdout output from the program execution. Screenshots that are not readable, or screenshot without "CS149 Spring 2021 PlayRoom from …" from the program, will receive 0 point for the entire homework.

b. (5 pts) Include code snippet (not paragraphs) and describe the flow of steps performed by a boy thread (or a girl thread) – including the exact calling sequence of synchronization constructs.

c. (60 pts) source code. Submit separate .c file.

Submit the following files as individual files (do not zip them together):

- `CS149_HW4_YourName_L3SID` (.pdf, .doc, or .docx), which includes
  - Q1: answers
  - Q2: a and b (note: c is in separate file)
- `playroom_YourName_L3SID.c` Your source code without binaries/executables. Ident your source code and include comments.

The ISA and/or instructor leave feedback to your homework as comments and/or annotated comment. To access annotated comment, click "view feedback" button. For details, see the following URL:
https://guides.instructure.com/m/4212/l/352349-how-do-i-view-annotation-feedback-comments-from-my-instructor-directly-in-my-assignment-submission

NOTE: the course requires you to use Linux VM even on Mac. There are known Pthread related issues on Mac. In particular, POSIX unnamed semaphore is **not** supported on Mac. If you still use Mac for HW4, you could use *named semaphore* or *Grand Central Dispatch* instead. For named semaphore, change the data type of each semaphore from "`sem_t`" to "`sem_t *`". For Grant Central Dispatch, change the data type of each semaphore from "`sem_t`" to "`dispatch_semaphore_t`". For details, see the following links:

http://man7.org/linux/man-pages/man7/sem_overview.7.html

https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man2/sem_open.2.html

http://stackoverflow.com/questions/1413785/sem-init-on-os-x

https://developer.apple.com/documentation/dispatch/dispatchsemaphore