Operating System Assignment#3

Cenhan Du

September 2020

1 Problem 3.1

1.1 a

Except for the original parent process, five extra processes are being created, as drawn below.

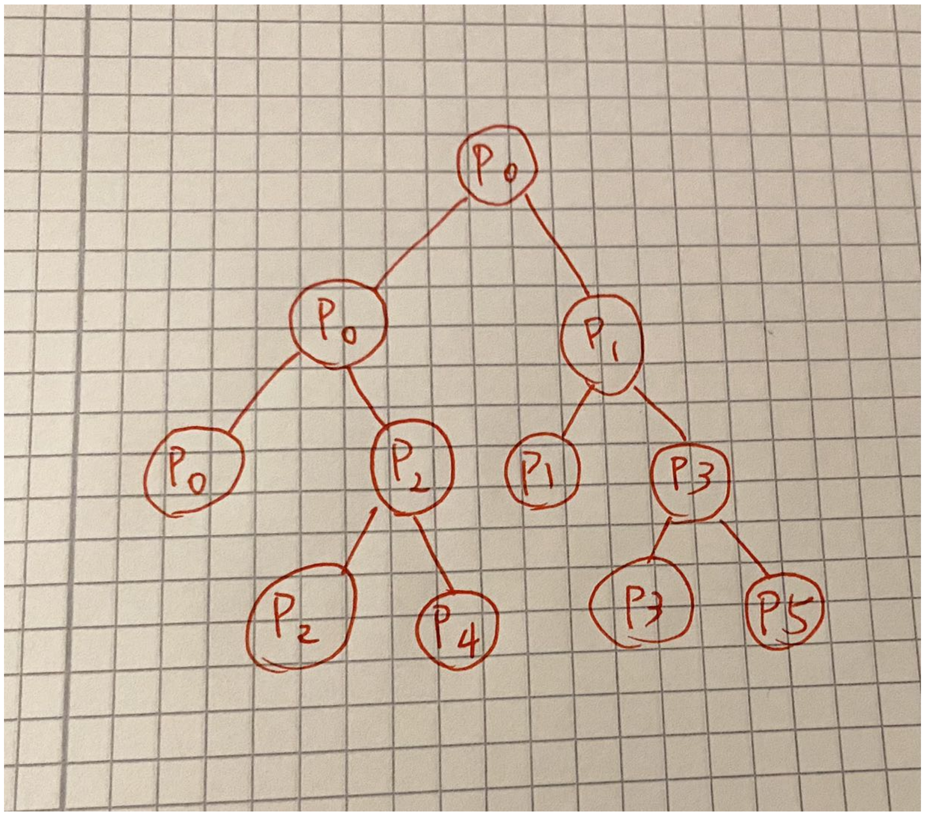

Figure 1: The process tree

1.2 b

The output produced by this program is
p0: x = 1
p1: x = 1
p2: x = 3
p3: x = 3
p4: x = 2
p5: x = 2

2 Problem 3.2

2.1 a

In this solution, at the last two lines of the reader function, the order is not correct. Lock is given back before the if clause.
If after one reader call up(&mutex) and before it notifies writers, another reader starts to read, 'readcount' will become one, which is not what the writers want. Because in that case the writer will overwrite things in the buffer which could be something the reader wants.

2.2 b

In this solution, in the last if clause, 'mutex' should be incremented after 'writer'. If after one reader calls up(&mutex) function, another reader comes in really fast and has already in the position of reading. The first reader then calls up(&writer), data would be overwritten by a writer.

2.3 c

In this solution, mutex is introduced in the writer body. This means writer can might go to sleep when mutex is 0.
At the end of one reader before it calls up(&mutex), if a writer tries to call down(&mutex), this writer will go to sleep before the reader increment mutex, which could be a very long time and thus fail the program.

3 Problem 3.3

```
shared object runners;
shared int relayRunners = 0;
semaphore chips = N, team = 1;
void runner(){
    down(&chips);
    run(&runners);
    up(&chips);
}
void relay(){
    while(relayRunners!=T){
        add_runner_to_team(&runners);
        relayRunners = relayRunners+1;
    }
    down(&team);
    relayRunners = 0;
    for(int i=0; i<T; i++)
        down(&chips);
    for(int i=0; i<T; i++){
        run(&runners);
        up(&chips);
    }
    up(&team);
}
```