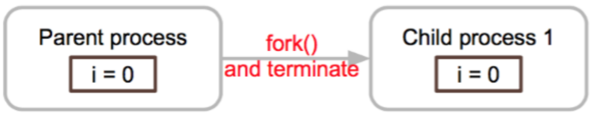
IT5002 Computer Systems and Applications
Tutorial 7 SOLUTIONS

1. How many processes are created in the following program? Include the original process created when the program is first run.

```
for(int i=0; i<10; i++) fork();
```

It is extremely tedious to work this out by hand. Fortunately there is an easier way. We start first with for(i=0; i<1; i++):

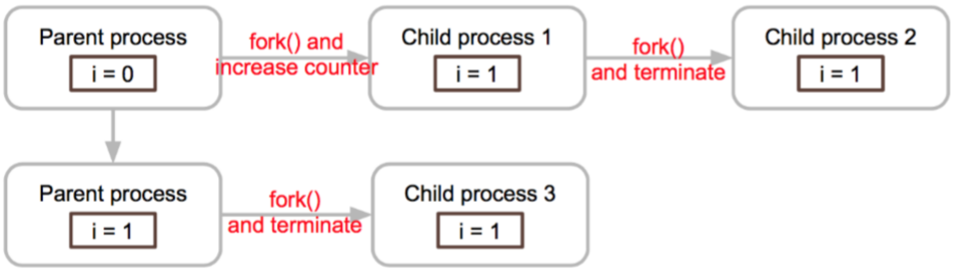
Analysis for (int i = 0; i < 1; i++):



Total number of processes: 2

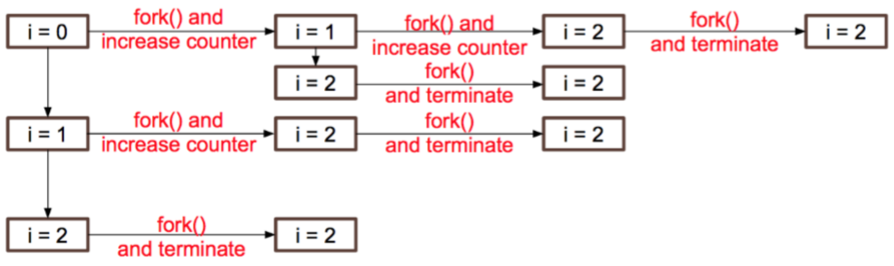
We do the same for for(i=0; i<2; i++):

Analysis for (int i = 0; i < 2; i++):



Total number of processes: 4

Analysis for (int i = 0; i < 3; i++):



Total number of processes: 8

```
We can see that the forks form essentially a binary
tree, and that for the general case n, we will have
2^n forks.
```

2. Consider a process P1. What state would P1 be in, in the following circumstances?

Situation	**P1's State**
The OS has just finished creating P1	Ready
P1 is pre-empted by another process.	Ready
P1 is waiting for I/O	Blocked
P1's I/O has just completed	Ready
P1 is currently being executed	Running
P1 is suspended for 2 seconds	Blocked

3. Suppose we have a fixed-priority operating system, where each process is given a priority, and when a higher priority process is ready, the OS pre-empts the current process and passes CPU control to the higher priority process.

   a. If there are 255 possible priority levels, how many ways can you assign priorities to *n* processes, if no two processes can have the same priority level?

   We can assign 255 priorities to the first process, 254 to the second process, ..., 255 – n priorities to the *n*th process. Total = $255 \times 254 \times \ldots (255 - n + 1)$

$$= \frac{255!}{(255-n)!}$$
$$= P(n, r)$$

b. We consider *periodic processes*. A periodic process is one that runs for C_i CPU cycles every P_i cycles.

Assuming that a process must finish running before the start of its next period, an intuitive way to assign priorities is by order of process periods; a process with a shorter period would be assigned a higher priority than one with a longer period; this is called *rate monotonic scheduling* or RMS.

So process P_1 might run for 1 cycle every 4 cycles, P_2 might run for 2 cycles every 6 CPU cycles, etc, giving a schedule that looks like this.

Clock Cycle	**Process**	**Ready to Run**
0	P1	P1, P2
1	P2	P1 runs first
2	P2	
3		
4	P1	P1
5		
6	P2	P2
7	P2	
8	P1	P1
9		
10		
11		
12	P1	P1, P2
13	P2	P2 delayed by one cycle because of P1
14	P2	
15		
16	P1	
…	…	

Suppose we sort our processes in ascending order of period, so that $\Pr(T_0) > PR(T_1) > PR(T_2) > \cdots > PR(T_n)$, where $\Pr(A) > \Pr(B)$ means that process Consider this algorithm:

1. Sort T by period of each task, if T is not already sorted. We will assume that T_1 has the shortest period, T_2 has the 2nd shortest, etc.
2. For each task $T_i \in T$, recursively compute $S_{i0}, S_{i1}, \ldots$ where:
    a. $S_{i,0} = \sum_{j=1}^{i} C_j$
    b. $S_{i,(x+1)} = C_i + \sum_{j=1}^{i-1} C_j \times \left\lceil \frac{S_{i,x}}{P_j} \right\rceil$

    Stop when $S_{i,(x+1)} = S_{i,x}$. Call this $S_{i,(x+1)}$ the final value $S_{i,F}$. I.e. let $S_{i,F} = S_{i,(x+1)}$ when the iteration terminates.

Prove that $S_{i,F}$ is the worst-case response time for process P_i. The response time is the time between the first time a process runs, and when it finishes running.

The sum of CPU times given by $S_{i,0}$ gives us the total response time of period *i* which includes the running times of all processes of higher priority.

However the OS may pre-empt process *i* when a higher priority process *j* becomes ready.

The number of times this happens is given by $\left\lceil \frac{S_{i,0}}{P_j} \right\rceil$, and if process j runs for C_j CPU cycles, the total delay caused by process *j* pre-empting process *i* is $C_j \times \lceil \frac{S_{i,0}}{P_j} \rceil$

If we repeat the analysis for all higher priority processes, the total delay on process i caused by all higher priority processes is:

$$\sum_{j=1}^{i-1} C_j \times \lceil \frac{S_{i,0}}{P_j} \rceil$$

Adding process i's own CPU time gives us:

$$S_{i,1} = C_i + \sum_{j=1}^{i-1} C_j \times \left\lceil \frac{S_{i,0}}{P_j} \right\rceil$$

If $S_{i,1} > S_{i,0}$ there may be further pre-emptions, in which case we recursively apply the formula above giving us:

$$S_{i,t+1} = C_i + \sum_{j=1}^{i-1} C_j \times \left\lceil \frac{S_{i,t}}{P_j} \right\rceil$$

When we $S_{i,t+1} = S_{i,t}$ this means that there are no further pre-emptions, and $S_{i,F} = S_{i,t+1}$ is thus the worst case response time for a process *i*.

4. Our system has one CPU and one I/O device.

   We have a set of 3 processes P1, P2 and P3 where P1 has the highest priority and P3 the lowest. The "execution profile" column is in the format Cw-IOx-Cy-IOz which means that the process runs for "w" cycles, then accesses IO for "x" cycles, then runs on the CPU for "y" cycles, and again performs IO for "z" cycles. All 3 processes are ready to run at time 0.

   When the I/O device is busy the next process will wait for it to be free before using it. If two processes are waiting for I/O, the higher priority process uses it first. However there is no pre-emption for I/O.

Process	**Execution Profile**
P1	C2-IO3-C3-IO1-C2-IO3-C3
P2	C1-IO2-C1
P3	C3

   a. Show the schedule for the three processes.

      (See next page)

   b. What is the response time for each process?

      P1: 17 cycles
      P2: 9 cycles
      P3: 12 cycles

   c. What is the CPU utilization? State your answer to two decimal places.

Utilization = $\frac{15}{17} \times 100$ = 88.24%

Cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CPU	P1	P1	P2	P3	P3	P1	P1	P1	P2	P1	P1	P3			P1	P1	P1
I/O			P1	P1	P1	P2	P2		P1			P1	P1	P1			