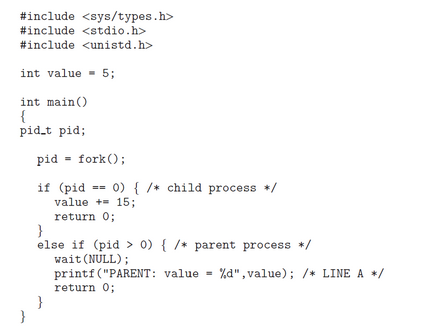
Due: Sun Oct 25, 2020 11:59pm

**Instructions**

Provide concise answers to the following questions. Submission must be a Word or PDF document and must contain the question text above your answer.

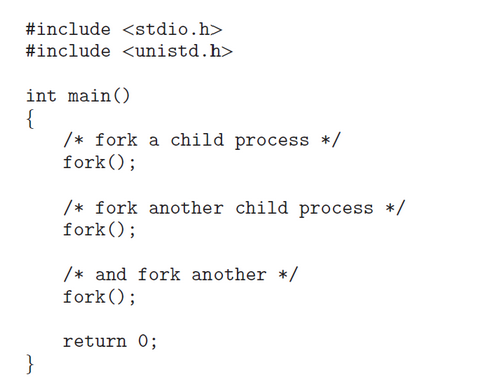
**Chapter 3 Questions**

1. Using the program shown below, explain what the output will be at LINE A.



PARENT: value = 5

2. Including the initial parent process, how many processes are created by the program shown below?



There are 8 processes that are created.

3. Original versions of Apple’s mobile iOS operating system provided no means of concurrent processing. Discuss three major complications that concurrent processing adds to an operating system.

a. It’s pretty complex if two or more processes need to use sharing global resources, when 2 processes access 1 resource(ex: file) concurrently, crash or lock can happen. So we need to handle these problems carefully.

b. It’s difficult to debug because the processes share the same resources.

c. Context switching between 2 processes can spend a lot of time.

4. When a process creates a new process using the fork() operation, which of the following states is shared between the parent process and the child process?

     a. Stack  
     b. Heap  
     c. Shared memory segments

c. Stack and heap aren’t shared between processes.

5. Describe the actions taken by a kernel to context-switch between processes.

The context switching between the processes is accomplished by the kernel.

In reply to the clock interrupt, OS stores the Program Counter value and the user Stack Pointer of the presently implementing process, then handovers charge to the kernel clock interrupt handler.

The clock interrupt handler holds back the remaining registers, along with other machine status such the status of the floating point registers in the Process Control Block of the process.

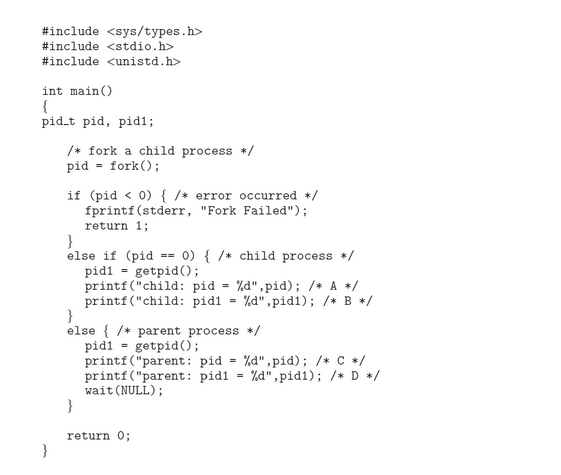
The OS calls upon the scheduler to decide the next process that has to be implemented.

The OS takes the status of the next process from its Process Control Block and fix up the registers. This restore task takes the processor back to the state in which the process was earlier interrupted, implementing in user code with user mode privileges.

6.  Explain the role of the init (or systemd) process on UNIX and Linux systems in regard to process termination.

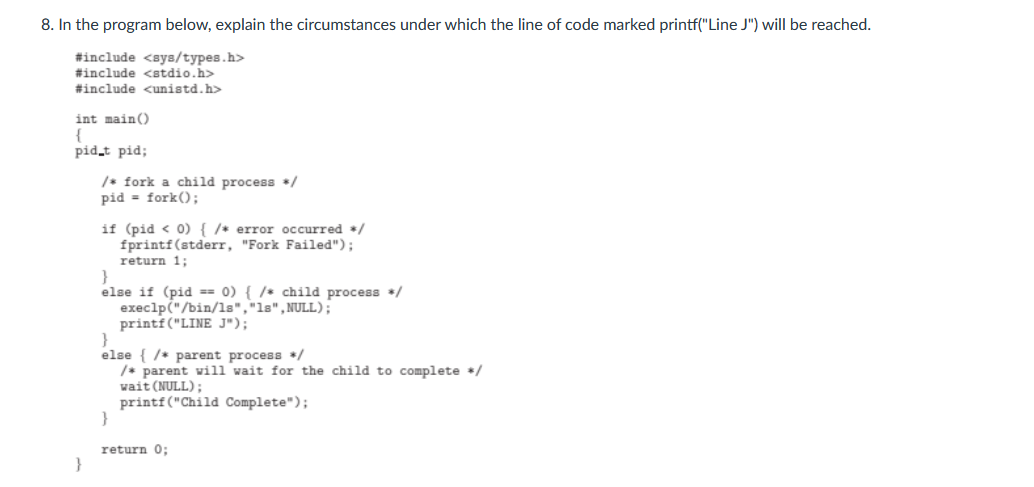
Init process is the first process started during booting, it’s also the parent of all processes. Its role is to create processes from script in file etc/inittab

7. Including the initial parent process, how many processes are created by the program shown below?



There are only 2 processes, parent and child.

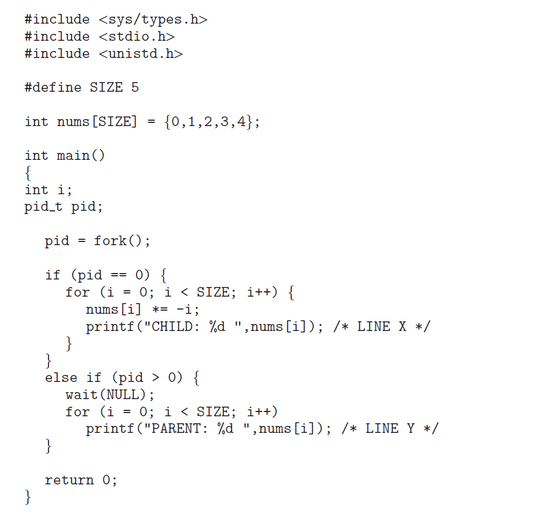
8. In the program above, explain the circumstances under which the line of code marked printf("Line J") will be reached.



The program never reach to *LINE J* since exec family replaces the process it is called from.

9. What are the benefits and the disadvantages of each of the following? Consider both the system level and the programmer level.  
     a. Synchronous and asynchronous communication   
Synchronous communication allows a rendezvous between the sender and receiver but disadvantage of a blocking send is that a rendezvous may not be required and the message could be delivered asynchronously. Otherwise, asynchronous → non-blocking  
     b. Automatic and explicit buffering  
Automatic buffering provides a queue with indefinite length, thus ensuring the sender will never have to block while waiting to copy a message. There are no specifications on how automatic buffering will be provided; one scheme may reserve sufficiently large memory where much of the memory is wasted. Explicit buffering specifies how large the buffer is. In this situation, the sender may be blocked while waiting for available space in the queue. However, it is less likely that memory will be wasted with explicit buffering.  
     c. Send by copy and send by reference  
Send by copy doesn’t allow the receiver to alter the state of the parameter. Send by reference does allow it.  
     d. Fixed-sized and variable-sized messages  
The implications of this are mostly related to buffering issues; with fixed-size messages, a buffer with a specific size can hold a known number of messages. The number of variable-sized messages that can be held by such a buffer is unknown.

10. What will be the output at lines X and Y in the program below?



Line X: CHILD: 0 CHILD: -1 CHILD: -4 CHILD: -9 CHILD: -16  
Line Y: PARENT: 0 PARENT: 1 PARENT: 2 PARENT: 3 PARENT: 4

**Chapter 4 Questions**

11.  Provide three programming examples in which multithreading provides better performance than a single-threaded solution.

Word program where a thread is used to monitor user input, another thread represents the running application.  
Image program uses multi thread to reduce the work time.  
A Web server that services each request in a separate thread.

12.  Using Amdahl’s Law, calculate the speedup gain of an application that has a 60 percent parallel component for (a) two processing cores and (b) four processing cores.

Amdahl’s Law: t = 1/(S+(1-S)/N)  
N: processing core number  
S: portion of application (=1-0.6=0.4)  
t: speedup gain

a. N=2  
=> t = 1/(0.4+(1-0.4)/2) = 1.43

b. N=4  
=> t = 1/(0.4+(1-0.4)/4) = 1.82

13. What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?

User-level threads are unknown by the kernel, whereas the kernel is aware of kernel threads.  
On systems using mapping, user threads are scheduled by the thread library and the kernel schedules kernel threads.  
Kernel threads need not be associated with a process whereas every user thread belongs to a process. Kernel threads are generally more expensive to maintain than user threads as they must be represented with a kernel data structure.

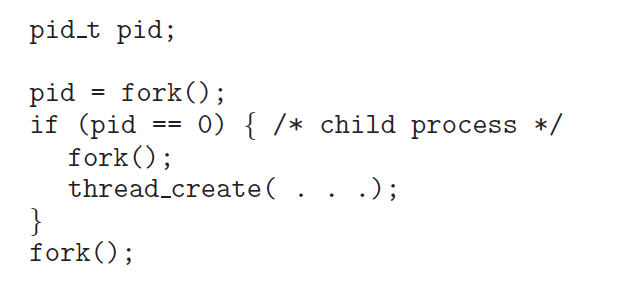
14. Describe the actions taken by a kernel to context-switch between kernel-level threads.15. What resources are used when a thread is created? How do they differ from those used when a process is created?

Context switching between kernel threads typically requires saving the value of the CPU registers from the thread being switched out and restoring the CPU registers of the new thread being scheduled.

15. In Chapter 3, we discussed Google’s Chrome browser and its practice of opening each new tab in a separate process. Would the same benefits have been achieved if, instead, Chrome had been designed to open each new tab in a separate thread? Explain.

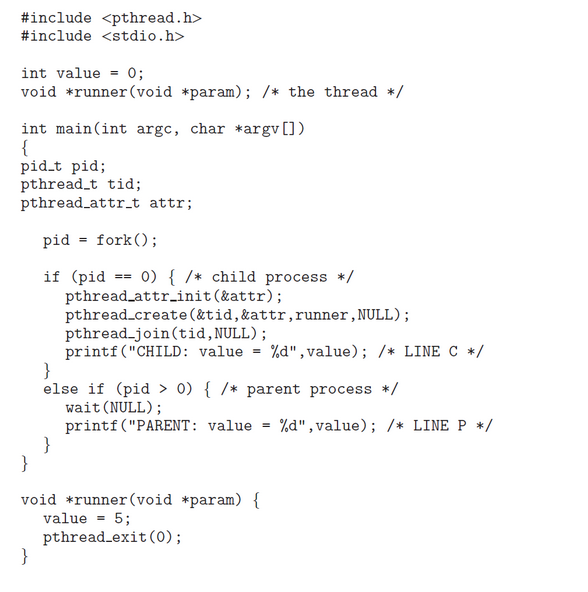
No benefit if Chrome opening each new tab in a separate thread because open each tab in process can avoid crash.

16. Consider the following code segment:



     a.  How many unique processes are created?  
 6 processes.  
  
     b.  How many unique threads are created?  
 3 threads.

17. The program shown below uses the Pthreads API. What would be the output from the program at LINE C and LINE P?



LINE C: CHILD: value = 5  
LINE P: PARENT: value = 0

**Chapter 5 Questions**

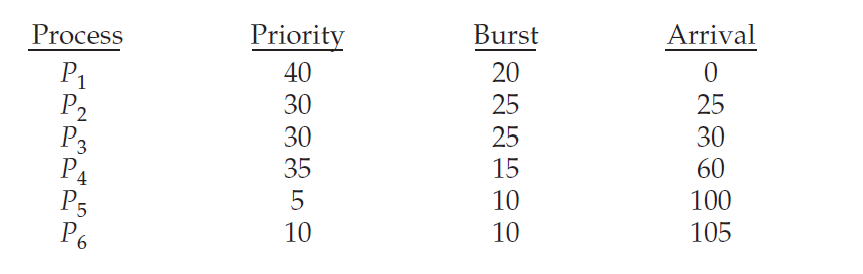
18. A CPU-scheduling algorithm determines an order for the execution of its scheduled processes. Given n processes to be scheduled on one processor, how many different schedules are possible? Give a formula in terms of n.

n!

19. Explain the difference between preemptive and non-preemptive scheduling.

Preemptive scheduling allows a process to be interrupted in the midst of its execution, taking the CPU away and allocating it to another process. Non-preemptive scheduling ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst.

20. The following processes are being scheduled using a preemptive, round robin scheduling algorithm.



Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed below, the system also has an idle task (which consumes no CPU resources and is identified as Pidle). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 10 units. If a process is preempted by a higher-priority  
process, the preempted process is placed at the end of the queue.  
     a. Show the scheduling order of the processes using a Gantt chart.

P1 –– idle –– P2 –– P3 –– P2 –– P3 –– P4 –– P4 –– P2 –– P3 –– idle –– P5 –– P6 –– P5  
0 10 20 25 35 45 55 60 70 75 80 90 100 115 120  
     b. What is the turnaround time for each process?

P1: 20-0=20, P2: 80-25=55, P3: 90-30=60, P4: 75-60=15, P5: 120-100=20, P6: 115-105=10  
     c. What is the waiting time for each process?

P1: 0, P2: 40, P3: 35, P4: 0, P5: 10, P6: 0  
     d. What is the CPU utilization rate?

105/120 = 85%

21. What advantage is there in having different time-quantum sizes at different levels of a multilevel queueing system?

Processes that need more frequent servicing, for instance, interactive processes such as editors, can be in a queue with a small time quantum. Processes with no need for frequent servicing can be in a queue with a larger quantum, requiring fewer context switches to complete the processing, and thus making more efficient use of the computer.

22. Many CPU-scheduling algorithms are parameterized. For example, the RR algorithm requires a parameter to indicate the time slice. Multilevel feedback queues require parameters to define the number of queues, the scheduling algorithms for each queue, the criteria used to move processes between queues, and so on. These algorithms are thus really sets of algorithms (for example, the set of RR algorithms for all time slices, and so on). One set of algorithms may include another (for example, the FCFS algorithm is the RR algorithm with an infinite time quantum). What (if any) relation holds between the following pairs of algorithm sets?  
     a. Priority and SJF  
The shortest job has the highest priority.  
     b. Multilevel feedback queues and FCFS  
The lowest level of MLFQ is FCFS.  
     c. Priority and FCFS  
FDFS gives the highest priority to the job having been existence the longest.  
    d. RR and SJF  
None.

23. Suppose that a CPU scheduling algorithm favors those processes that have used the least processor time in the recent past. Why will this algorithm favor I/O-bound programs and yet not permanently starve CPU-bound programs?

It will favor the I/O-bound programs because of the relatively short CPU burst request by them; however, the CPU-bound programs will not starve because the I/O-bound programs will relinquish the CPU relatively often to do their I/O.

24. The traditional UNIX scheduler enforces an inverse relationship between priority numbers and priorities: the higher the number, the lower the priority. The scheduler recalculates process priorities once per second using the following function:

Priority = (recent CPU usage / 2) + base

where base = 60 and recent CPU usage refers to a value indicating how often a process has used the CPU since priorities were last recalculated.

Assume that recent CPU usage for process P1 is 40, for process P2 is 18, and for process P3 is 10. What will be the new priorities for these three processes when priorities are recalculated? Based on this information, does the traditional UNIX scheduler raise or lower the relative priority of a CPU-bound process?

The priorities assigned to the processes are (40/2)+60=80, (18/2)+60=69 , and (10/2)+60=65 respectively. The scheduler lowers the relative priority of CPU-bound processes.

25. Using the Windows scheduling algorithm, determine the numeric priority of each of the following threads.  
     a. A thread in the REALTIME PRIORITY CLASS with a relative priority of NORMAL  
24

     b. A thread in the ABOVE NORMAL PRIORITY CLASS with a relative priority of HIGHEST  
12  
     c. A thread in the BELOW NORMAL PRIORITY CLASS with a relative priority of ABOVE NORMAL  
7