King Saud University
Department of Computer Science
CSC227: Operating Systems
Tutorial – Chapter 3: Processes

Q 1) What is a process?
Executable program with its own address space and process control block.

Q 2) How does a child process differ from its parent?
It uses separate address space and separate stack pointer from the parent pointer.

Q 3) What is a process control block (PCB)?
A process control block is a data structure used by an operating system to manage processes.

Q 4) Is a PCB found all operating systems?
Process control blocks are found in all modern operating systems.

Q 5) Is PCB in all operating systems the same?
The structure of PCBs will be different on each operating system though.

Q 6) What does process context switching involves?
Changing Process Control Block structures between an old and a new process

Q 7) Context switching operation represents a pure overhead in the OS process management. What does it means?
It means that no other useful work can be done during the context switch operation.

Q 8) List at least FIVE other kinds of data that will be stored in a PCB that is not shown in PCB diagram.
PCB contains a great deal of information about the process, including:

- The scheduling priority
- Information about the user and groups associated with this process
- Pointer to the parent process PCB
- Pointer to a list of child process PCBs
- Pointers to the process's data and machine code in memory
- Pointers to open files and other resources held by the process

Q 9) Describe the actions a kernel takes to context switch between processes.
In general, the operating system must save the state of the currently running process and restore the state of the process scheduled to be run next. Saving the state of a process typically includes the values of all the CPU registers in addition to memory allocation. Context switches must also perform many architecture-specific operations, including flushing data and instruction caches.

Q 10) A computer has multiple register sets. Describe the actions of a context switch if the new context is already loaded into one of the register sets. What else must happen if the new context is in memory rather than a register set, and all the register sets are in use?
The CPU current-register-set pointer is changed to point to the set containing the new context, which takes very little time. If the context is in memory, one of the contexts in a register set must be chosen and moved to memory, and the new context must be loaded from memory into the set. This process takes a little more time than on systems with one set of registers, depending on how a replacement victim is selected.

Q 11) What is the advantage of restricting a child process to a subset of the parent's process?
The processes will limit within the resource allocated to parent and thus will not exhaust system resources.

Q 12) List the reasons when parent may terminate the execution of its child.
The child has exceeded its usage of system resources, the task assigned to the child is no longer required, the parent is exiting, and the OS does not allow a child to continue if its parent terminates.)

Q 13) How does provision of multiple registers help in context switch?
If multiple registers are provided, then the context switch simply requires changing the pointer to the current register set. Of course, if there are more process than the number of registers available, the OS must copy the register data to and from memory.

Q 14) Mention the three different queues a process may be place in by the OS before it completes execution.
Job queue, ready queue, and waiting queue (device queue).

Q 15) In a context switch, the OS made changes to some fields in the PCB of the current running process. Mention two fields in the PCB that will be updated with new information.
Register and state.

Q 16) Discuss the different states that a process can exist in at any given time.
The possible states of a process are: new, running, waiting, ready, and terminated. The process is created while it is in the new state. In the running or waiting state, the process is executing or waiting for an event to occur, respectively. The ready state occurs when the process is ready and waiting to be assigned to a processor and should not be confused with the waiting state mentioned earlier. After the process is finished executing its code, it enters the termination state.

Q 17) What are the differences between a short-term and long-term scheduler?
The primary distinction between the two schedulers lies in the frequency of execution. The short-term scheduler is designed to frequently select a new process for the CPU, at least once every 100 milliseconds. Because of the short time between executions, the short-term scheduler must be fast. The long-term scheduler executes much less frequently; minutes may separate the creation of one new process and the next. The long-term scheduler controls the degree of multiprogramming. Because of the longer interval between executions, the long-term scheduler can afford to take more time to decide which process should be selected for execution.

Q 18) Discuss in detail the producer/consumer problem.
In the producer/consumer problem, a producer creates items and a consumer uses them. The items are stored in a shared buffer, which can be infinite or of a limited size. The producer and consumer must synchronize on the buffer contents so that items are not lost or consumed more than once. If the buffer is empty, the consumer must wait for the producer to create a new item. If a finite buffer is full, the producer must wait for the consumer to use an item.

Q 19) List three different situations lead to the following direct transitions between states:

a) running state to ready state: Interrupt, time-out (end of time slice), arrival of a higher priority process, sleep system call.
b) waiting state to ready state: End of I/O, parent becomes ready after child complete execution
c) running state to waiting state: Request for I/O, process forks a child, wait for an event, wait system call.

Q 20) Describe the actions taken when a user process starts to execute a write() system call.

1) The write() call is a privileged instruction that traps to the kernel.
2) The kernel saves the user process context.
3) The kernel sets up registers in the I/O controller and transfers the data to be written to the controller.
4) The kernel selects a (different) ready process and switches context to that process.
5) When output is complete, the I/O device generates an interrupt.
6) The interrupt service routine saves the current CPU context, does some accounting, and moves the writing user process into the ready queue.
7) The kernel either 1) switches context to the original process executing the system call, or 2) returns control to the second process, or 3) selects and switches context to an entirely different process.

Q 21) How many times the message will be printed?

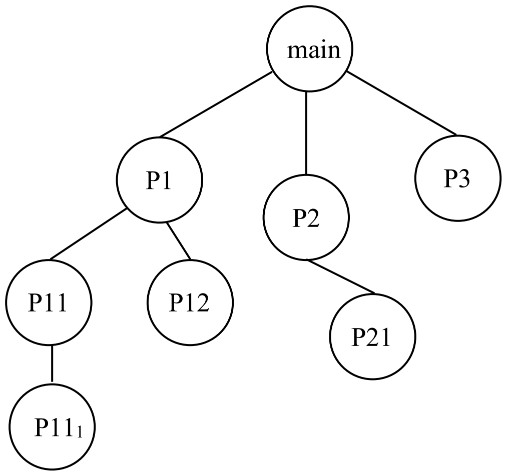
```
main()
{
      fork();
      printf("Hello world");
}
```

Answer: 2 times. Child is created at fork() and every line after **fork** will be executed twice once by parent (i.e. printf("Hello world");) and once by child (again printf("Hello world");)

Q 22) Consider the following C language program, what are the possible outputs?

```
#include <stdio.h>
int num;
int main(){
      num = 1;
      fork();
      num = num + 1;
      printf(num);
}
```

Answer: It will print value of num two times; one from parent and one from child.

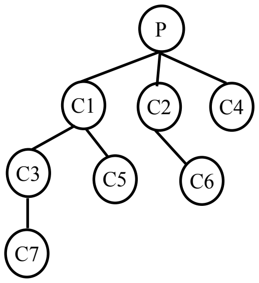


Q 23) How many processes will be created when the following program is executed (including the parent process)?

```
#include <stdio.h>
#include <unistd.h>
int main() {
      fork();
      fork();
      fork();
      return 0;
}
```

Answer: 8 processes.

Explanation: The parent process (P) will execute the first fork which will result in a new child process (C1).
Both P and C1 will then execute the second fork, hence P will have another child process (C2), and C1 will have a child process (C3).
Then, P, C1, C2, C3 will execute the third fork, which will result in C4 as a child of P, C5 as a child of C1, C6 as a child of C2 and C7 as a child of C3.
Since there are no more forks, the created processes are P, and C1-7 = 8 processes.



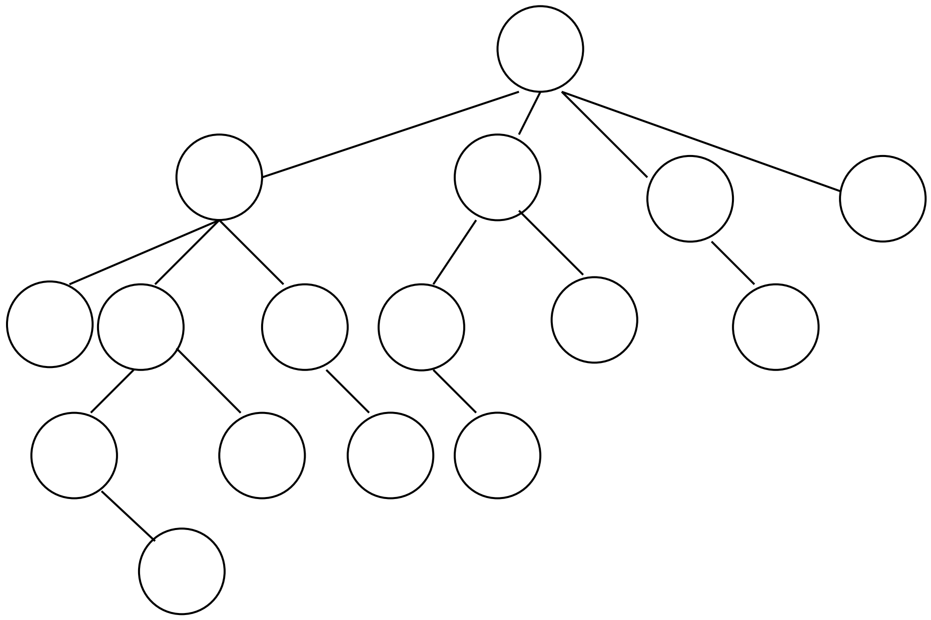
Q 24) How many times does the program below print Hello?

```
#include <stdio.h>
int main()
{
      fork();
      fork();
      fork();
      printf( "Hello\n" );
}
```

Answer: 8 times.

Q 25) How many processes are created when the following piece of code is executed? Draw the process tree for the processes thus created.

```
int main() {
      int i;
      for (i=0; i<4; i++)
            fork();
      return 1;
}
```



Answer: 16 processes (main + 15 forked processes).

Q 26) Consider the following C program:

```
#include<sys/types.h>
#include<stdio.h>
#include<unistd.h>
int v=5;
int main() {
   pid_t pid;
   pid=fork();
   if (pid==0) {
      v+=15;
      printf("value=%d\n",v);
      return 0;
   } else if (pid>0) {
        wait(NULL);
        printf("value=%d\n",v);
        return 0;
   }
}
```

a) What is the output of the parent process?
Answer: value = 5

Explanation: The value of v will remain 5 as the child process will increase the value of its copy of the variable v. The parent's copy of the variable v will remain unchanged.

b) What is the output of the child process?
Answer: value=20
Explanation: The child process will increase the value of v by 15. The new value of v will be 20.

Q 27) What will be printed on the screen after executing the following program?

```
#include<sys/types.h>
#include<stdio.h>
#include<unistd.h>
int v=5;
int main() {
   pid_t pid;
   pid=fork();
   if (pid==0) {
      v+=15;
      printf("value=%d\n",v);
      return 0;
   } else if (pid>0) {
        wait(NULL);
        printf("value=%d\n",v);
        return 0;
   }
}
```

Answer:

```
value=20
value=5
```

Explanation: The child process will print before the parent process as the parent process has to wait for the termination of the child process before it executes the print statement which comes after the `wait()` system call.