CMPT 300
In-class activity 1 Solution

1. Give one example of fork() in common applications.
   - When we double-click on an icon, a program (new process) opens in a new window.
   - Running a sub-process on a shell creates a new command with a fork.

2. When a process is created using fork(), the _____ is not inherited by the child process.
   Process id
   parent's list of file descriptors

3. Generally, we call fork () and exec (). Let us assume that you write a shell, but your code first calls exec () and then calls fork () like the following code. Does it work? Explain.

```
1   shell(…) {
2
3   .. ..
4   exec(command, args);
5   fork();
6   .. ...
7   }
```

Does not work. Shell's address space is entirely replaced with the new command; therefore, the shell will terminate once the command is terminated.

4. Explain how a system call is different from a typical procedure call.

For system calls, there will be a mode switch from user mode to kernel mode.

5. Assume that the root process has pid = 1000; what can you say about the pid of a child?
   a) It will be greater than 1001
   b) Any number greater than 1000
   c) It Could be any random number. We cannot predict.

   Give reason.

   It could be any random number. We cannot predict. Since there is a limit on the maximum number of pids, they will be reused after a certain time. So, a child's pid does not always need to be greater than the parent's.

6. Let us say we put the parent process to sleep, and meanwhile, the child process exits. What will happen in this scenario?
   The child will become a zombie process (A process that has finished the execution but still has an entry in the process table to report to its parent process)

7. Why should we call wait(NULL) in the parent process? What if we didn't call wait() in the parent process?
   wait(NULL) block parent process until any of its children exits.

The child might become an orphan process if it takes longer than the parent to exit (A process whose parent process no longer exists i.e. either finished or terminated without waiting for its child process to terminate, is called an orphan process.)

8. Consider the code given in Q10. Suppose we modify our global variable in the parent process to 100 and print it in the child process what will be the output (select one or more)?
   a) If parent modifies before child print global_var will be 100
   b) If child prints before parent modifies global_var will be 50
   c) global_var will be 100 always
   d) global_var will be 50 always
   Give reason.

   Global_var will be 50 always.
   Since both processes will have their own copy of global_var, so no matter what the parent does with global_var, it will be 50 in the child process.

9. Fill in the correct boolean expression to complete the fragments.
   if(_____)
   printf("I am not an orphan child \n");

   getppid() != -1

10. Consider the given code

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/wait.h>

int global_var = 50;

int main(int argc,char *argv[]){
    pid_t pid = fork();
    if(pid==0)
    {
        printf("I'm child...\n");
    }
    else
    {
        printf("I'm parent...\n");
        wait(NULL);
    }
    execv("./dummy",argv);
}
```

```c
//dummy.c
#include <stdio.h>

int main(int argc,char* argv[])
{
    printf("executing dummy...\n");
}
```

   a) Which process will execute the print statement first? Parent process or the child process?Since the process will run concurrently, the one scheduled on the CPU will execute first.

   b) What will be the output of the program?
   Some sample outputs

```
I'm parent...
I' m child...
executing dummy...
executing dummy...
```

```
I' m child...
I'm parent...
executing dummy...
executing dummy...
```

c) What will be the output if we move execv() just after calling fork?

```
executing dummy...
executing dummy...
```

d) What would happen if we move execv() before calling fork? What will be the output then?

```
executing dummy...
```

e) What will be output if we put one more execv after the current execv in main.c ?
Below are sample outputs.

```
I' m child...
I'm parent...
executing dummy...
executing dummy...
```

```
I'm parent...
I' m child...
executing dummy...
executing dummy...
```

11. Suppose we have 3 different files, same as dummy.c as mentioned below:

Dummy1.c printf("executing Dummy1\n");
Dummy2.c printf("executing Dummy2\n");
Dummy3.c printf("executing Dummy3\n");

Now consider the following psedocode to execute these dummy files with execv.

```
main ()
{
  for (i from 1 to 3)
    {
      Pid = fork ();
      if (child)
    {
      execv ()
        //call ith  dummy file here
    }
    }
}
```

What will be the output of the following program here?
Executing Dummy1
Executing Dummy2
Executing Dummy3
(order may be different if an interrupt occurs while executing or because of scheduling)

12. How many times does the following program print "Hello World"? Draw a simple tree diagram to show the parent-child hierarchy of the spawned processes.

```
int main ()
{
  int i;
  for (i = 0; i < 3; i++)
    fork ();
  printf ("test\n");
  return 0;
}
```

Answer: "test" is printed 8 times.4