Some solutions to exercises on Processes

Most of the exercises are programs. So please code the programs and run them to find your solutions. I only provide solutions to questions 7 and 8

1. Including the initial parent process, how many processes are created by the following program? Construct a tree that shows the parent-child relationship among processes.

```
#include <stdio.h>
#include <unistd.h>
int main()
{
      /* fork a child process */
         fork();
      /* fork another child process */
         fork();
      /* and fork another */
         fork();
         return 0;
}
```

2. In the following program, what will be the output at Line A? Explain briefly.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int value = 5;
int main()
{
pid_t pid;
    pid = fork();
    if (pid == 0) { /* child process */
       value += 15;
       return 0;
    }
    else if ( pid > 0){ /* parent process */
       wait(NULL);
       printf(PARENT: value =%d? value);/* Line A */
       return 0;
    }
}
```

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

```
int main()
int i;
pid_t pid;
for (i=0;i<5;i++)
   pid = fork();
return 0;
```

4. Using the program below, what output will be at lines X and Y?

```
#define SIZE 7

int nums[SIZE] = {7,6,5,4,3,2,1}

int main()
   int i; pid_t pid;
   pid = fork();
   if (pid == 0) {
      for (i=0;i < SIZE; i++){
         nums[i] = nums[i] * -2*i;
         printf("CHILD: %d ", nums[i]); /*LINE X */
      }
   }
   else if (pid > 0) {
      wait(NULL);
      for (i=0;i<SIZE;i++)
         printf("PARENT: %d ", nums[i]); /*LINE Y */
   }
   return 0;
```

5. The following recurrence takes a positive integer n and eventually reach 1:

$$n = \begin{cases} n/2 & \text{if } n \text{ is even} \\ 3 \times n + 1 & \text{if } n \text{ is odd} \end{cases}$$

For example, if $n = 35$, the sequence is

$$35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1$$

Write a C or C++ program using the fork() system call that generates this sequence in the child process. The starting number will be provided from the standard input. Have the parent invoke the wait() call to wait for the child process to complete before exiting the program.

6. Give three conditions that cause switching from user mode to kernel mode.

7. What is a system call? How does a system call differ from a call to a method or a function in a software library? What kinds of services can be obtained through system calls?

   sol: a request for a service from the operating system's kernel. Systems involve an interruption of the process making the call and a transfer to the OS, switching form user mode to kernel, etc. All this is time consuming. Services provided by the OS.

8. Comparing system calls vs. procedure calls. The simple test program below compares the cost of a simple procedure call to a simple system call. Compile and run this program.

   (a) Run your experiment on two different hardware architectures. Report your results for each architecture.

   (b) Explain the difference (if any) between the time required by your simple procedure call and simple system call by discussing what work each call must do.

   sol: A system call is expected to be significantly more expensive than a procedure call (provided that both perform very little actual computation). A system call involves the following actions, which do not occur during a simple procedure call, and thus entails a high overhead: A context switch A trap to a specific location in the interrupt vector Control passes to a service routine, which runs in 'monitor' mode The monitor determines what system call has occurred Monitor verifies that the parameters passed are correct and legal

```
#include <sys/time.h>
#include <unistd.h>
#include <assert.h>
#include <stdio.h>

int foo(){
  return(10);
}

long nanosec(struct timeval t){ /* Calculate nanoseconds in a timeval structure */
  return((t.tv_sec*1000000+t.tv_usec)*1000);
}

main(){
  int i,j,res;
  long N_iterations=1000000; /* A million iterations */
  float avgTimeSysCall, avgTimeFuncCall;
  struct timeval t1, t2;

  /* Find average time for System call */
  res=gettimeofday(&t1,NULL); assert(res==0);
  for (i=0;i<N_iterations; i++){
    j=getpid();
  }
```

```
  res=gettimeofday(&t2,NULL);   assert(res==0);
  avgTimeSysCall = (nanosec(t2) - nanosec(t1))/(N_iterations*1.0);

  /* Find average time for Function call */
  res=gettimeofday(&t1,NULL);  assert(res==0);
  for (i=0;i<N_iterations; i++){
    j=foo();
  }
  res=gettimeofday(&t2,NULL);   assert(res==0);
  avgTimeFuncCall = (nanosec(t2) - nanosec(t1))/(N_iterations*1.0);


  printf("Average time for System call getpid : %f\n",avgTimeSysCall);
  printf("Average time for Function call : %f\n",avgTimeFuncCall);
}
```

9. Using the program below and the command "execl", make the child process to execute the command ps -f which is in the directory /bin

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main(void) {
pid_t childpid;
pid_t mypid = getpid();
printf("I am parent with PID = %ld\n", (long)mypid);
childpid = fork();
if (childpid == 0) { /* child code */
  printf("I am child with PID = %ld\n", (long)getpid());
  execl();
return 1;
}
if (childpid != wait(NULL)) { /* parent code */
perror("Parent failed to wait due to signal or error");
return 1;
}
return 0;
}
```

10. Here are two versions of the code that generates a chain of processes. Run these two versions, observe the outputs and explain why the two outputs are not the same.

    (a)
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
/*create a chain of processes. Take an integer as argument which is the
length of the chain.*/
```

```
int main (int argc, char *argv[]) {
pid_t childpid = 0;
int i, n;
n = atoi(argv[1]);
 for (i = 0; i < n; i++)
   if (childpid = fork()) break;
fprintf(stderr, "i:%d process ID:%ld child ID:%ld, parent ID:%ld\n",i,
(long)getpid(), (long)childpid,(long)getppid());
wait(NULL);
 return 0;
}
```

(b)
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
/*create a chain of processes. Take an integer as argument which is the
length of the chain.*/
int main (int argc, char *argv[]) {
pid_t childpid = 0;
int i, n;
n = atoi(argv[1]);
 for (i = 0; i < n; i++)
   if (childpid = fork()) break;
 wait(NULL);
 fprintf(stderr, "i:%d process ID:%ld child ID:%ld, parent ID:%ld\n",
i, (long)getpid(), (long)childpid,(long)getppid());
return 0;
}
```

11. Here there is again a new version of the program that generates a chain of processes. There is two possible interpretations about what this program is doing:

    (a) It forces each child process to replace itself by a new binary

    (b) it forces each process including the master to replace itself by a new binary

    Does the program below will work? If yes, which of the above two interpretations is correct? You may observe zombies, can you explain why this happens? Notice the last process to be created does something different from the others, could you explain why?

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main (int argc, char *argv[]) {
pid_t childpid = 0;
```

```
int i, n;
n = atoi(argv[1]);
 for (i = 0; i < n; i++)
   if (childpid = fork()){
execl("/bin/ps", "ps", "-f", NULL);
break;
}
 wait(NULL);
 fprintf(stderr, "i:%d process ID:%ld child ID:%ld, parent ID:%ld\n",
i, (long)getpid(), (long)childpid,(long)getppid());
return 0;
}
```

12. Using the execl() command, change the program below such that the child processes will execute the command "date". How many processes will execute the fprintf() command, which one execute it and why?

```
#include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <sys/wait.h>
/*create a fan of processes (a tree of level 1). */
int main (int argc, char *argv[]) {
pid_t childpid = 0;
int i, n;
n = atoi(argv[1]);
for (i = 1; i < n; i++)
  if ((childpid = fork()) <= 0) break;
fprintf(stderr, "i:%d process ID:%ld child ID:%ld, parent ID:%ld\n",
i, (long)getpid(), (long)childpid,(long)getppid());
 wait(NULL);
return 0;
}
```

13. The program below generates a tree of processes. Do you think it is possible for one of these processes to print the total number of processes in the tree?

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
/* Create a multilevel tree of processes*/
int main (int argc, char *argv[]) {
pid_t childpid = 0;
 int i,j, n;
n = atoi(argv[1]);
 for (i = 1; i < n; i++){
  childpid = fork();
  if (childpid != 0)fprintf(stderr, "tree: %d i:%d process ID:%ld parent ID:%ld chi
```

```
    tree,i, (long)getpid(), (long)getppid(), (long)childpid);
 }
 sleep(10);
return 0;
}
```