操作系统作业 4

姓名，学号

1. Why is the separation of mechanism and policy desirable?

   Answer: Mechanism and policy must be separate to ensure that systems are easy to modify. No two system installations are the same, so each installation may want to tune the operating system to suit its needs. With mechanism and policy separate, the policy may be changed at will while the mechanism stays unchanged. This arrangement provides a more flexible system.

2. What are the two models of interprocess communication? What are the strengths and weaknesses of the two approaches?

   Answer: The two models of interprocess communication are message-passing model and the shared-memory model. Message passing is useful for exchanging smaller amounts of data, because no conflicts need be avoided. It is also easier to implement than is shared memory for intercomputer communication. Sharedmemory allows maximum speed and convenience of communication, since it can be done at memory transfer speeds when it takes place within a computer. However, this method compromises on protection and synchronization between the processes sharing memory.

3. Including the initial parent process, how many processes are created by the program shown in Figure 1?

```
#include <stdio.h>
#include <unistd.h>

int main()
{
   int i;

   for (i = 0; i < 4; i++)
      fork();

   return 0;
}
```

Figure 1: Program for Question 3.

Answer: 16 processes are created.

Hint: You may include printf() statements to better understand how many processes have been created.

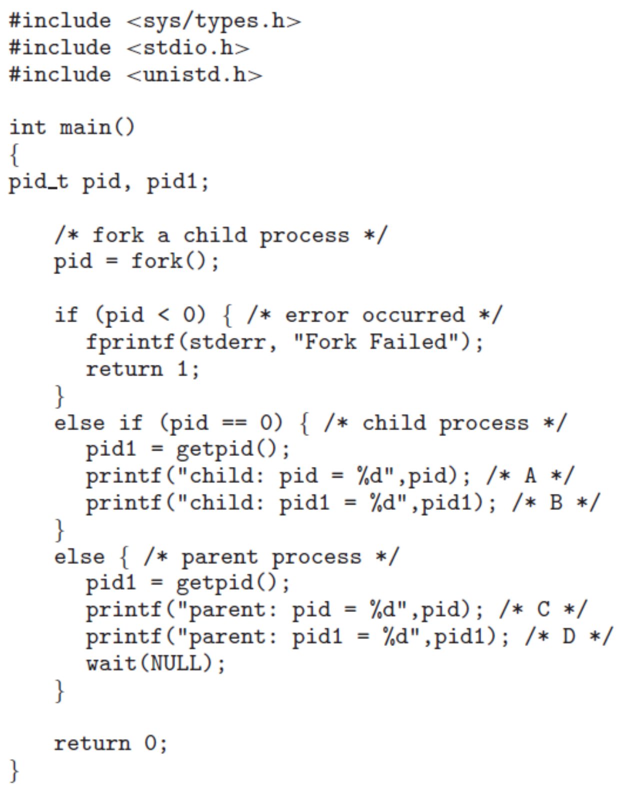
4. Explain the circumstances under which the line of code marked printf (“LINE J”) in Figure 2 will be reached

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

   /* fork a child process */
   pid = fork();

   if (pid < 0) { /* error occurred */
     fprintf(stderr, "Fork Failed");
     return 1;
   }
   else if (pid == 0) { /* child process */
     execlp("/bin/ls","ls",NULL);
     printf("LINE J");
   }
   else { /* parent process */
     /* parent will wait for the child to complete */
     wait(NULL);
     printf("Child Complete");
   }

   return 0;
}
```

Figure 2: Program for Question 4.

Answer: The call to exec() replaces the address space of the process with the program specified as the parameter to exec(). If the call to exec() succeeds, the new program is now running and control from the call to exec() never returns. In this scenario, the line printf("Line J"); would never be performed. However, if an error occurs in the call to exec(), the function returns control and therefor the line printf("Line J"); would be performed.

5. Using the program in Figure 3, identify the values of pid at lines A, B, C, and D. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.)



```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid, pid1;

   /* fork a child process */
   pid = fork();

   if (pid < 0) { /* error occurred */
     fprintf(stderr, "Fork Failed");
     return 1;
   }
   else if (pid == 0) { /* child process */
     pid1 = getpid();
     printf("child: pid = %d",pid); /* A */
     printf("child: pid1 = %d",pid1); /* B */
   }
   else { /* parent process */
     pid1 = getpid();
     printf("parent: pid = %d",pid); /* C */
     printf("parent: pid1 = %d",pid1); /* D */
     wait(NULL);
   }

   return 0;
}
```

Figure 3: Program for Question 5.

Answer: A = 0, B = 2603, C = 2603, D = 2600

6. Using the program shown in Figure 4, explain what the output will be at lines X and Y.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

#define SIZE 5

int nums[SIZE] = {0,1,2,3,4};

int main()
{
int i;
pid_t pid;

   pid = fork();

   if (pid == 0) {
      for (i = 0; i < SIZE; i++) {
         nums[i] *= -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;
}
```

Figure 4: Program for Question 6.

Answer: Because the child is a copy of the parent, any changes the child makes will occur in its copy of the data and won't be reflected in the parent. As a result, the values output by the child at line X are 0, -1, -4, -9, -16. The values output by the parent at line Y are 0, 1, 2, 3, 4

7. Which of the following components of program state are shared across threads in a multithreaded process?

   a. Register values

   b. Heap memory

   c. Global variables

   d. Stack memory

Answer: The threads of a multithreaded process share heap memory and global variables. Each thread has its separate set of register values and a separate stack.

8. A system with two dual-core processors has four processors available for scheduling. A CPU-intensive application is running on this system. All input is performed at program start-up, when a single file must be opened. Similarly, all output is performed just before the program terminates, when the program results must be written to a single file. Between startup and termination, the program is entirely CPU-bound. Your task is to improve the performance of this application by multithreading it. The application runs on a system that uses the one-to-one threading model.

   (1) How many threads will you create to perform the input and output? Explain your reason.

   (2) How many threads will you create for the CPU-intensive portion of the

application? Explain your reason.

Answer:

(1) It only makes sense to create as many threads as there are blocking system calls, as the threads will be spent blocking. Creating additional threads provides no benefit. Thus, it makes sense to create a single thread for input and a single thread for output.

(2) Four. There should be asmany threads as there are processing cores. Fewer would be a waste of processing resources, and any number > 4 would be unable to run.

9. Consider the following code segment:

```
pid t pid;

pid = fork();

if (pid == 0) { /* child process */

    fork();

    thread create( . . .);

}

fork();
```

a. How many unique processes are created?

b. How many unique threads are created?

Answer: There are six processes and two threads.

10.The program shown in Figure 5 uses the Pthreads API. What would be the output from the program at `LINE C` and `LINE P`?

```
#include <pthread.h>
#include <stdio.h>

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
pid_t pid;
pthread_t tid;
pthread_attr_t attr;

  pid = fork();

  if (pid == 0) { /* child process */
    pthread_attr_init(&attr);
    pthread_create(&tid,&attr,runner,NULL);
    pthread_join(tid,NULL);
    printf("CHILD: value = %d",value); /* LINE C */
  }
  else if (pid > 0) { /* parent process */
    wait(NULL);
    printf("PARENT: value = %d",value); /* LINE P */
  }
}

void *runner(void *param) {
  value = 5;
  pthread_exit(0);
}
```

Figure 5: C program for Question 10.

Answer: Output at LINE C is 5. Output at LINE P is 0.