Chapter 2 exercises

1 Using the program shown in the following figure, explain what the output will be at LINE A .

```c
#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;
    }
}
```

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

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

3 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

4 Describe the differences among short-term, medium-term, and long-term scheduling.

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

6 Including the initial parent process, how many processes are created by the program shown bellow?

```c
#include < stdio.h >
#include < unistd.h >
int main() {
  int i;
  for (i = 0; i < 4; i++)
    fork();
```

```
  return 0;
}
```

7 Explain the circumstances under which which the line of code marked printf("LINE J") in following code 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;
}
```

8 Using the following program, 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;
}
```

9 Give an example of a situation in which ordinary pipes are more suitable than named pipes and an example of a situation in which named pipes are more suitable than ordinary pipes.

10 Using the program shown below, 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;
}
```

11 What resources are used when a thread is created? How do they differ from those used when a process is created?

12 Provide two programming examples in which multithreading does not provide better performance than a single-threaded solution.

13 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

14 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?

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

```
#include <pthread.h>
#include <stdio.h>
#include <types.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);
}
```

16 We mentioned that disabling interrupts frequently can affect the system's clock. Explain why this can occur and how such effects can be minimized.

17 What is the meaning of the term busy waiting? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your answer.

18 Explain why spinlocks are not appropriate for single-processor systems yet are often used in multiprocessor systems.

19 Show that, if the wait() and signal() semaphore operations are not executed atomically, then mutual exclusion may be violated.

20 Race conditions are possible in many computer systems. Consider a banking system that maintains an account balance with two functions: deposit(amount) and withdraw(amount) . These two functions are passed the amount that is to be deposited or withdrawn from the bank account balance. Assume that a husband and wife share a bank account. Concurrently, the husband calls the withdraw() function and the wife calls deposit() . Describe how a race condition is possible and what might be done to prevent the race condition from occurring.

21 Explain why interrupts are not appropriate for implementing synchronization primitives in multiprocessor systems.