Assignment-8

September 30, 2021

[1]Farhin Mashiat Mayabee [2]MD. Mehedi Hasan

1. Write the output of the following code-

```c
#include <stdio.h>
#include <unistd.h>
int main()
{
    pid_t status;
    status = fork();
    if (status == 0)
    {
        printf("PID = %d & Parent PID = %d\n", getpid(),getppid());
    }
    else
    {
        status = fork();
        if (status == 0)
        {
            printf("PID = %d & Parent PID = %d\nstion",getpid(), getppid());
        }
        else
        {
            printf("PID = %d & Parent PID = %d\n",getpid(), getppid());
        }
    }
    return 0;
}
```

N.B: You can use any relatable ID.

Ans: PID = 20010 Parent PID = 7338
PID = 20011 Parent PID = 20010
PID = 20012 Parent PID = 20010

2. How many times would ‘F1’, ‘F2’ and ‘M’ be printed in this following code-

```c
#include <stdio.h>
#include <unistd.h>
void function1(){
    if(fork())
    printf("F1\n");
}
void function2(){
    printf("F2\n");
}
int main()
{
    fork();
    function1();
    fork();
    function2();
    printf("M\n");
    return 0;
}
```

Ans: F1- 2
F2- 8
M - 8

3. Watch the following code. In this code, Outputs are not stable. So, solve this problem so that data is not inconsistent.

```
#define _GNU_SOURCE
#include <sys/ipc.h>
#include <stdio.h>
#include <sys/shm.h>
#include <string.h>
#include <sched.h>
#include <unistd.h>
#include <stdlib.h>
#include <semaphore.h>
#include <sys/wait.h>
int create_shared_memory(){
  key_t key;
  int shmID;
key=ftok("/home/cse/OperatingSystem/Code/Semaphore_Processes.c",'a');
  shmID = shmget(key, 1024, IPC_CREAT | 0666);
  return shmID;
}
void write_to_shm(int x){
  int shmID = create_shared_memory();
  char *str = shmat(shmID, NULL, 0);
  char num[100];
  sprintf(num, "%d", x);
  strcpy(str, num);
  shmdt(str);
}
int read_from_shm(){
  int shmID = create_shared_memory();
  char *str = shmat(shmID, NULL, 0);
  int num = (int) strtol(str, (char **)NULL, 10);
  shmdt(str);
  return num;
}
int main(){
  pid_t childPID;
  int a, b, x;
  x = 10;
  write_to_shm(x);
  childPID = fork();
  if (childPID == 0){
    a = read_from_shm();
    printf("x: %d in Child [Core: %d] Before Addition\n", a, sched_getcpu());
    a = a + 1;
    write_to_shm(a);
    b = read_from_shm();
    printf("x: %d in Child [Core: %d] After Addition\n", b, sched_getcpu());

  }else{
    a = read_from_shm();
    printf("x: %d in Parent [Core: %d] Before Subtraction\n", a, sched_getcpu());
    a = a - 1;
    write_to_shm(a);
    b = read_from_shm();
    printf("x: %d in Parent [Core: %d] After Subtraction\n", b, sched_getcpu());
    waitpid(childPID, NULL, 0);
  }
  return 0;
}
```

4. What would be the last value of v?

```
#include <stdio.h>
#include <semaphore.h>
#include <unistd.h>
void fun(){
    fork();
}
int main(){
    int v;
    sem_t key;
    sem_init(&key, 0, 6);
    sem_getvalue(&key, &v);
    printf("The initial value of the semaphore is %d\n", v);
    sem_post(&key);
    sem_wait(&key);
    fork();
    sem_wait(&key);
    sem_wait(&key);
    sem_wait(&key);
    sem_post(&key);
    fun();
    sem_getvalue(&key, &v);
    printf("The value of the semaphore after the wait is %d\n", value);
    return 0;
}
```

Ans: 8

5. Discuss briefly how ftok() works.

Ans: a) 'path' must refer to an existing, accessible file.
b) 'id' must be nonzero.
The ftok() function does not guarantee unique key generation. The key is created by combining:
a) the given id byte, i.e., 8 bits,
b) the lower 16 bits of the inode number,
c) the lower 8 bits of the device number into a 32-bit result.
The occurrence of key duplication is very rare.

6. Write some uses of pipes.

Ans: Pipe is a connection between two processes. Pipes are useful for communication between related processes(inter-process communication).
a) Pipe is one-way communication.We can use pipe to write from one process and read from another process.
b) The pipe can be used by the creating process, as well as all its child processes, for reading and writing. One process can write to this "virtual file" or pipe and another related process can read from it. c) We can use pipe to send information from one process to another which are not necessarily on the same directory.

7. What are the advantages and disadvantages of shared memory?

   Ans: Advantages of Shared Memory Programming-
   1.Data sharing between processes is both rapid and uniform because of the proximity of memory to CPUs.
   2.Insignificant process communication overhead.
   3.Global address space offers a user-friendly programming perspective to memory.
   4.No need to specify explicitly the communication of data between processes.
   5.More intuitive and easier to learn.

   Disadvantages-
   1.Difficult to manage data locality.
   2.User is responsible for specifying synchronization e.g. locks.
   3.Not portable.
   4.Scalability is limited by the number of access pathways to memory.

8. Write the general ways how the deadlock can be handled.

   Ans: Deadlock can be handled by one of three ways:
   1. Pretend: Pretend deadlock will never happen. used by most operating systems, including UNIX
   2. Prevent or Avoid: Ensure that the system will never enter a deadlock state.
   3. Detect and Recover: Let system to enter a deadlock state and then take a step to recover.

9. What is difference between deadlock prevention and deadlock avoidance?

   Ans: Deadlock prevention is a set of methods for ensuring that at least one of necessary conditions for deadlock cannot hold. Deadlock avoidance requires that operating system be given, in advance, additional information concerning which resources a process will request and use during its lifetime.

10. Write advantages and disadvantages of user level thread.

    Ans: Some of the advantages of user-level threads are as follows
    1. User-level threads are easier and faster to create than kernel-level threads. They can also be more easily managed.
    2. User-level threads can be run on any operating system.
    3. There are no kernel mode privileges required for thread switching in user-level threads.
    Disadvantages of User-Level Threads
    Some of the disadvantages of user-level threads are as follows
    1. Multithreaded applications in user-level threads cannot use multiprocessing to their advantage.
    2. The entire process is blocked if one user-level thread performs blocking operation.

11. Single threaded process vs Multi threaded process.

12. How to prevent race condition?

    Ans: To prevent the race conditions from occurring, we can lock shared variables, so that only one thread at a time has access to the shared variable.

13. What are the differences between User level and kernel level thread?

User Level	Kernel Level
User thread are implemented by users.	kernel threads are implemented by OS.
Implementation of User threads is easy.	Implementation of Kernel thread is complicated.
Takes less time on context switching	Takes more time on context switching

Table 1: Caption

14. Describe the three common ways to establish relation between user threads and kernel threads.

    Ans: There is a relationship between user-level threads and kernel threads. Three common ways to establish relation between user threads and kernel threads:
    1. Many-to-One Model
    2. One-to-One Model
    3. Many-to-Many Model

    Many-to-One Model Many user-level threads are mapped to one kernel thread. Only one thread can access the kernel at a time. The entire process will block if a thread makes a blocking system call.

    One-to-One Model Each user thread is mapped to a kernel thread. Another thread is allowed to run when a thread makes a blocking system call. Creating a user thread requires creating the corresponding kernel thread.

    Many-to-Many Model

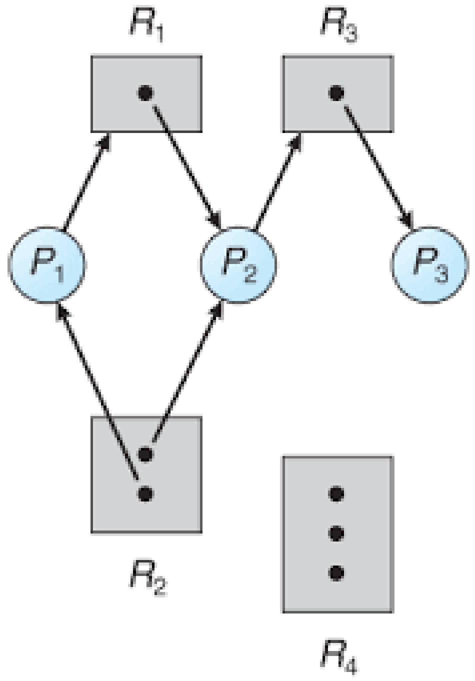
    Many user-level threads are mapped to a smaller or equal number of kernel threads. When a thread performs a blocking system call, the kernel can schedule another thread of execution.

15. Why is the behaviour of thread always unpredictable?

    Ans: Thread behaviour is unpredictable because its execution depends on the thread scheduler. And every thread scheduler is different from the other. So in different platforms thread acts differently.

16. Explain deadlock with resource allocation graph.

    Ans: Deadlock is a situation where two or multiple processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process. Here is a deadlock shown in resource allocation graph.


R_1
R_3
P_1
P_2
P_3
R_2
R_4