Assignment 3

Ramana Srivats S

Anna University

1. Which signals are triggered, when the following actions are performed.
    1. user press ctrl+C SIGINT
    2. kill() system call is invoked

        The kill() system call is used to send any signal to any specified process. This means that depending on the signal which is passed as the argument, the same signal will be triggered
    3. CPU tried to execute an illegal instruction SIGILL
    4. When the program access the unassigned memory SIGSEGV
2. List the gdb command for the following operations
    1. To run the current executable file

        run
    2. To create breakpoints at

        break [line number]
    3. To resume execution once after breakpoint

        continue
    4. To clear break point created for a function

        clear [function name]
    5. Print the parameters of the function in the backtrace

        info args

3. Guess the output for the following program.

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

int main()
{
if (fork() && (!fork())) {
if (fork() || fork()) {
fork();
}
}
printf("2");
return 0;
}
```

OUTPUT

2 2 2 2 2 2 2

4. Guess the output for the following program.

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

int main()
{
   if (fork()) {
      if (!fork()) {
         fork();
         printf("1 ");
      }
      else {
         printf("2 ");
      }
   }
```

```
    else {
        printf("3 ");
    }
    printf("4 ");
    return 0;
}
```

OUTPUT

2 4 1 4 1 4 3 4

5. Create two thread functions to print hello and world separately and create threads for each and execute them one after other in C

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
void *printHello()
{
    printf("Hello ");
    return NULL;
}
void *printWorld()
{
    printf("World");
}
int main()
{
    pthread_t thread1,thread2;
    pthread_create(&thread1, NULL, printHello, NULL);
    pthread_join(thread1, NULL);
    pthread_create(&thread2, NULL, printWorld, NULL);
    pthread_join(thread2, NULL);

}
```

6. How to avoid Race conditions and deadlocks?

Race conditions and deadlocks can be avoided by the usage mutex, semaphores or locks. Race conditions happen when two processes try to access and change the data present at the same time which might cause some discrepancies and dead lock happens when two processes hold each other's required resources and both of them wait for the other process to complete which will not happen. By using mutex or semaphores each process will get to access shared resources one at a time (i.e.) when one process is accessing the other process have to wait.

7. What is the difference between exec and fork?

   fork()

   - fork system call creates a new process called child process which is copy of the process in which it is called.
   - Both parent and child processes execute simultaneously.

   exec()

   - exec function replaces the current process with a new process in the same process space.
   - The control only returns back to the previous process when there is an exec error.

8. What is the difference between process and threads.

   Process

   - A process is an instance of a program being executed.
   - Each process is independent of each other and does not work on shared resources.
   - Communication between two processes requires more time.
   - If one of the processes get blocked, the other processes will continue its execution.

   Threads

   - Thread is a segment of a process (i.e.) a division within the process.
   - Threads are interdependent and share computer's resources with each other.
   - Communication between threads take less time when compared to processes.

- If any user level thread gets blocked, all the peer threads will also get blocked.

9. Write a C program to demonstrate the use of Mutexes in threads synchronization.

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

static pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
long long i = 0;
void *countTo10000()
{
   while (1)
   {
      pthread_mutex_lock(&mutex);
      if ( i >= 10000)
      {
         pthread_mutex_unlock(&mutex);
         return NULL;
      }
      ++i;
      pthread_mutex_unlock(&mutex);
      printf("%lld\n",i);
   }
}

int main()
{
   int j = 0;

   pthread_t *threads = malloc(sizeof(pthread_t)*15);

   for (j=0 ; j < 15; j++)
   {
      pthread_create(&threads[i], NULL, countTo10000, NULL);
```

```
    }

    for (j=0 ; j < 15; j++)
    {
       pthread_join(threads[i], NULL);
    }
    printf("%lld\n",i);
    return 0;
}
```