8/10

COSC 350 System Software (Mini Test #4)

11/05/21

Name: Jung An

1. (4 pt.) Write a complete C program that creates a child, which runs forever. The parent process and child process must run concurrently. The child process keep printing, "Mom! I am immortal!" . The parent process print "This is your mom!" 10 times and then try to terminate the child process by sending SIGUSR1 to child. When the child process get the signal from parent, child response with a message "My said "go away", then terminate. Once the child is terminated, parent process will say "Sorry My Son" three times. **Do not use global variable, do not use wait() or waitpid()**

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

void childkill()
{ printf("My Said /" go away /");
  exit(0);
}
void parentkill()
{ printf("Sorry my son");
  exit(0);
}
int main(
{ pid_t pid;
  pid=fork();
 if(pid==0){
       while(1){
         printf("Mom! I am immortal!");
         signal(SIGUSR1, childkill())
         }
   }
   else{
     int i;
     for(i=0; i<10; i++){
       Printf("This is your mom!");
     }

         kill(pid, SIGUSR1);
         signal(SIGCHILD, parentkill());
         pause()

    }
 return 0;
}
```

do you need sigchild?

2. (4 pt.) Write a following syntax error free compliable C program.

   Two threads will be created. Three threads are running concurrently.
   Each thread will continuously print "This is original thread", "This is the first thread" or "This is the second thread". For each print, each thread will sleep one second.
   Original program runs forever until recognize the first thread's termination.
   After the second thread print 10 times, it tried to terminate the first thread. Then, the second tread print "The second thread job is done" and terminate itself.
   When the first thread recognizes the second thread's trial, the first thread will print "The first thread job is done" and terminate. When the original program recognizes the first thread's termination, it prints "The original thread job is done" and terminate itself.

   Do not use pthread_join() function for synchronization. Do not use signal for thread's termination!

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

void thread1(void *t)
{ while (1){
      printf("This is the first thread\n");
      if(pthread_testcancel()==0)
      {  pthread_cancel(t);
         printf("The first thread job is done");
       pthread_exit(NULL);
      }
void thread2()
{  int i;
   for(i=0;i<10;i++){
      printf("This is the second thread\n");
   }
   pthread_cancel(threads[0]);
   printf("The second thread job is done");
   pthread_exit(NULL);
}

int main()
{
   int NUMBER=2;
   pthread_t threads[NUMBER], orig;
   int rc;
   orig = pthread_self();

   rc=pthread_create(&threads[0], NULL, thread1, (void*)orig);
   rc=pthread_create(&threads[1], NULL, thread2, NULL);
   while(1)
   { printf("This is original thread\n");
      if(pthread_testcancel()==0)
      { printf("The original thread job is done");
        pthread_exit(NULL);
      }
   }
   pthread_exit(NULL);
   return 0;
}
```

How do you know?

yes and pass pointer for the first thread id

-15

3. 2 pt.) What will be displayed by following program?

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int gValue = 10;

void main()
{
   char *sId;
   int    iStack = 20;

   pid_t pID = fork();
   if (pID == 0)
   {
      sId = "Child Process: ";
      gValue++;
      iStack++;
    }
    else if (pID < 0)
    {
        printf( "Failed to fork\n");
        exit(1);
    }
    else
    {
      wait(NULL);
      sId = "Parent Process:";
      gValue=gValue+5;
      iStack=  iStack+5;
    }
    printf("%s\n", sId);
    printf(" Global variable: %d\n", gValue);
    printf( " Stack variable: %d\n", iStack);
}
```

Child Process:
Global variable: 11
Stack variable: 21

Parent Process:
Global variable: 12
Stack variable: 21