Daniyar Nazarbayev

Bilal Jama

Operating Systems – Coursework 2.

Q1 – B

Q1 – A is in Q3 section

```
#include <stdio.h>   /* printf, stderr, fprintf */
#include <sys/types.h> /* pid_t */
#include <unistd.h>  /* _exit, fork */
#include <stdlib.h>  /* exit */
#include <errno.h>   /* errno */
#include <sys/wait.h> /*wait */

int main(void)
{
 pid_t  pid;
 int fd[2];
 int done = 0;
 int status;

 if (pipe(fd) == -1)
   return (1);

 pid = fork();
   if (pid == 0) {

   close (fd[1]);
   while(done == 0){
    read(fd[0], &done, sizeof(done));
          }

   printf("I am the child process.\n");
   printf("The child is done \n");
```

```
    exit(0);
    }
    else {
    close(fd[0]);
    printf("This is the parent process.\n" );

    done = 1;
    write(fd[1], &done, sizeof(done));

    wait(&status);
    printf("The parent is done \n");
    exit(1);
  }
}
```

Q2 (A, B, C)

run with -lpthread

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

void *print_message_function( void *ptr );
void *print_message_function_2( void *ptr );

 /*
 "I am the parent thread"
 "The parent is done"
 "I am the child thread"
 "The child is done"
 */

struct thread_data{
        pthread_t* thread_id;
  char* first_message;
  char* second_message;
  int make_child;
  struct thread_data * child_data;
};

int main(int argc, char** argv)
{
if (argc == 2){
```

```
        //printf("%s \n", argv[1]);

        int parent;
    pthread_t thread1, thread2;

    struct thread_data * data1 = malloc(sizeof(struct thread_data));
    struct thread_data * data2 = malloc(sizeof(struct thread_data));
    data1->thread_id = &thread1;
        data1->first_message = "I am the parent thread";
        data1->second_message = "The parent is done";
        data1->make_child = 1;
        data1->child_data = data2;

        data2->thread_id = &thread2;
        data2->first_message = "I am the child thread";
        data2->second_message = "The child is done";
        data2->make_child = 0;
        data2->child_data = NULL;

        if(strcmp(argv[1], "a") == 0 || strcmp(argv[1], "c") == 0){
                parent = pthread_create( &thread1, NULL, print_message_function, (void*) data1);
        } else if(strcmp(argv[1], "b") == 0){
                parent = pthread_create( &thread1, NULL, print_message_function_2, (void*) data1);
        }
        // wait till thread 1 is done
        pthread_join(thread1, NULL);

        pthread_exit(NULL);
}
```

```
exit(0);
}

void *print_message_function( void *ptr )
{
        struct thread_data * temp = ptr;
   printf("%s \n", temp->first_message);
   printf("%s \n", temp->second_message);

   if(temp->make_child == 1){
        pthread_create( &*temp->child_data->thread_id, NULL, print_message_function, (void*) temp->child_data);
        pthread_join(*temp->child_data->thread_id, NULL);
        }

        pthread_exit(NULL);
}

void *print_message_function_2( void *ptr )
{
        struct thread_data * temp = ptr;
   printf("%s \n", temp->first_message);

        if(temp->make_child == 1){
        pthread_create( &*temp->child_data->thread_id, NULL, print_message_function, (void*) temp->child_data);
        pthread_join(*temp->child_data->thread_id, NULL);
        }
```

```
    printf("%s \n", temp->second_message);


            pthread_exit(NULL);
}
```

Q3 (and Q1 A)

```
#include <stdio.h>   /* printf, stderr, fprintf */
#include <sys/types.h> /* pid_t */
#include <unistd.h>  /* _exit, fork */
#include <stdlib.h>  /* exit */
#include <errno.h>   /* errno */
#include <sys/wait.h> /*wait */

int main(void)
{
        int fd[2];
        int done = 0;
        pid_t  pid;
        int status;

 if (pipe(fd) == -1)
   return (1);


pid = fork();
   if (pid == 0) {
                close (fd[1]);
                while(done == 0){
                        read(fd[0], &done, sizeof(done));
                }

      printf("I am the child process.\n");
      printf("The child is done \n");
```

```
        exit(0);
    }
    else {
    close(fd[0]);
    printf("This is the parent process.\n" );
    printf("The parent is done \n");

    done = 1;
    write(fd[1], &done, sizeof(done));

    wait(&status);

    exit(1);
  }

}
```

Q4

According to this answer on stackoverflow, only the thread that makes the fork call is replicated.

https://stackoverflow.com/questions/1073954/fork-and-existing-threads