**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>