CECS 326-01

Operating Systems

Dylan Dang (ID 026158052)

Assignment 2

Due Date: 10/04/2022

Submission Date: 10/04/2022

Program Description

The purpose of this assignment is to incorporate the fork(), exec(), and wait(), msgget(), msgctl(), msgsnd(), and msgrcv() system calls, within our three C++ programs named *master*.cc, sender.*cc*, and *receiver.cc*.

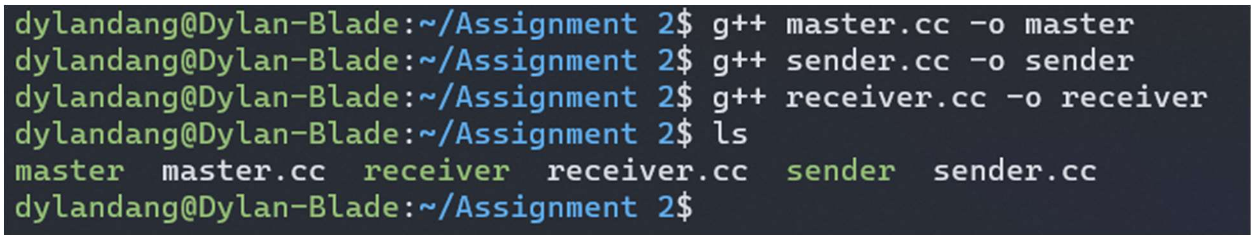
- Fork() allows a new process to be created and consists of a copy of the address space of the original process. This mechanism allows the parent process to communicate easily with its child process.
- Exec() replaces the process's memory space with a new program. The exec() system call loads a binary file into memory and starts its execution.
- Wait() moves the parent process itself off the ready queue until the termination of the child.
- Msgget() creates a message queue in which the key_t type and flag field return the identifier associated with the value of the key argument.
- Msgctl() can be used on existing message queues via control operations such as remove.
- Msgsnd() identifies the message queue by the identifier that was passed in via a commandline argument and then appends a copy of the user's inputted message that is pointed to by into the message buffer.
- Msgrcv() takes the message out of the message buffer by identifying the message queue using the identifier argument. The message is then placed into the buffer.

The *master.cc* program is designed to make use of the fork(), exec(), and wait() system calls to create two child processes, sender and receiver, and control the child processes' execution. The program starts off with identifying itself. The program then obtains a message queue from the kernel using the msgget() system call, followed by creating two child processes, with one to execute 'sender' and one to execute 'receiver', and pass to them the message queue id. The *master* process should output its process ID, the message queue ID, the process IDs of the *sender* and *receiver* processes it has created, then waits for both child processes to terminate. Then *master* will remove the message queue using the msgctl() system call and exit.

The *sender.cc* program is designed to first output a message to identify itself and show the message queue id it obtained from the exec() system call that invokes its execution from the *master* process. The program then prompts the user for a message and then send that message to the receiver via the message queue created by *master* by calling msgsnd(). The program will wait for an acknowledgement from *receiver* of the receipt of message and then terminate.

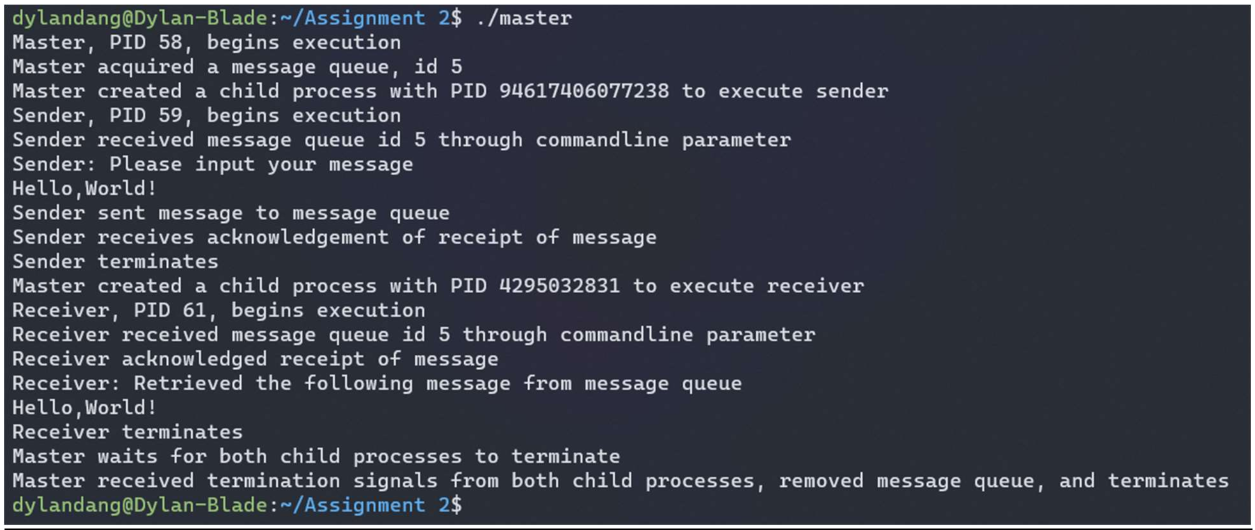
The *receiver.cc* program is designed to first output a message to identify itself and show the message queue id it obtained from the exec() system call that invokes its execution from the *master* process. *Receiver* retrieves the message from the message queue via the msgrcv() system call, sends an acknowledgement to *sender*, output the message on screen, and exit.

master.cc, sender.cc, receiver.cc compilation screenshot:



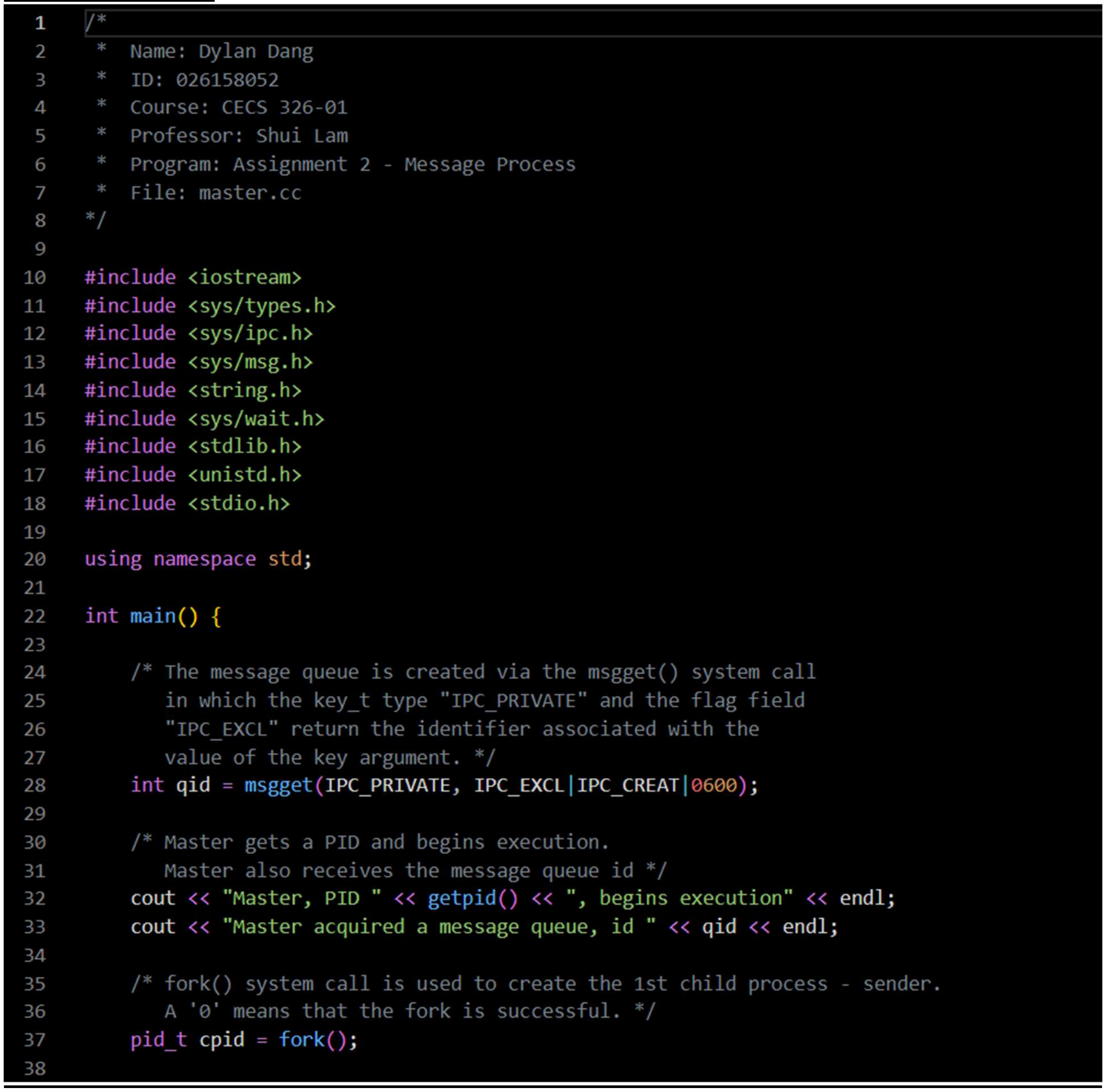
```
dylandang@Dylan-Blade:~/Assignment 2$ g++ master.cc -o master
dylandang@Dylan-Blade:~/Assignment 2$ g++ sender.cc -o sender
dylandang@Dylan-Blade:~/Assignment 2$ g++ receiver.cc -o receiver
dylandang@Dylan-Blade:~/Assignment 2$ ls
master  master.cc  receiver  receiver.cc  sender  sender.cc
dylandang@Dylan-Blade:~/Assignment 2$
```

running master executable:

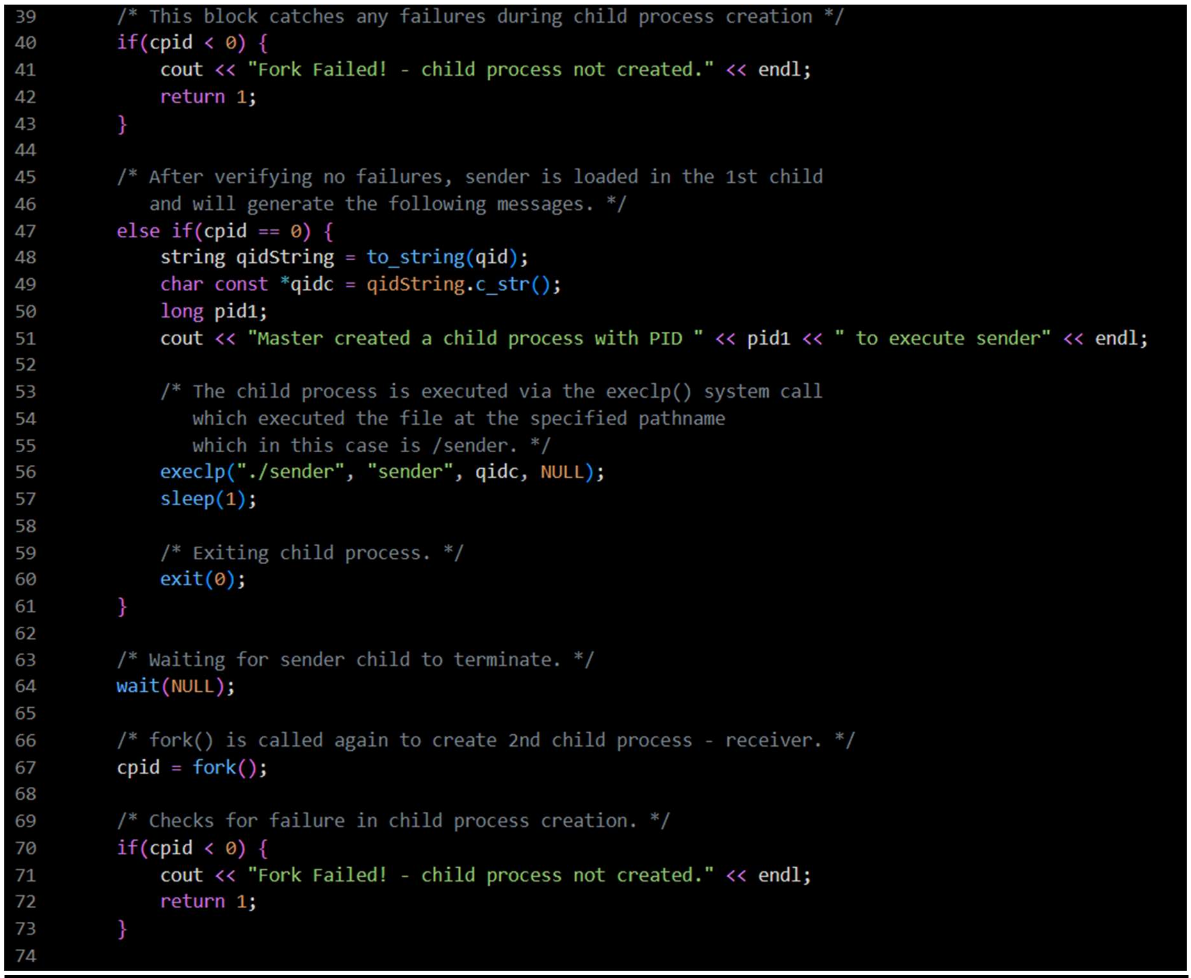


```
dylandang@Dylan-Blade:~/Assignment 2$ ./master
Master, PID 58, begins execution
Master acquired a message queue, id 5
Master created a child process with PID 94617406077238 to execute sender
Sender, PID 59, begins execution
Sender received message queue id 5 through commandline parameter
Sender: Please input your message
Hello,World!
Sender sent message to message queue
Sender receives acknowledgement of receipt of message
Sender terminates
Master created a child process with PID 4295032831 to execute receiver
Receiver, PID 61, begins execution
Receiver received message queue id 5 through commandline parameter
Receiver acknowledged receipt of message
Receiver: Retrieved the following message from message queue
Hello,World!
Receiver terminates
Master waits for both child processes to terminate
Master received termination signals from both child processes, removed message queue, and terminates
dylandang@Dylan-Blade:~/Assignment 2$
```

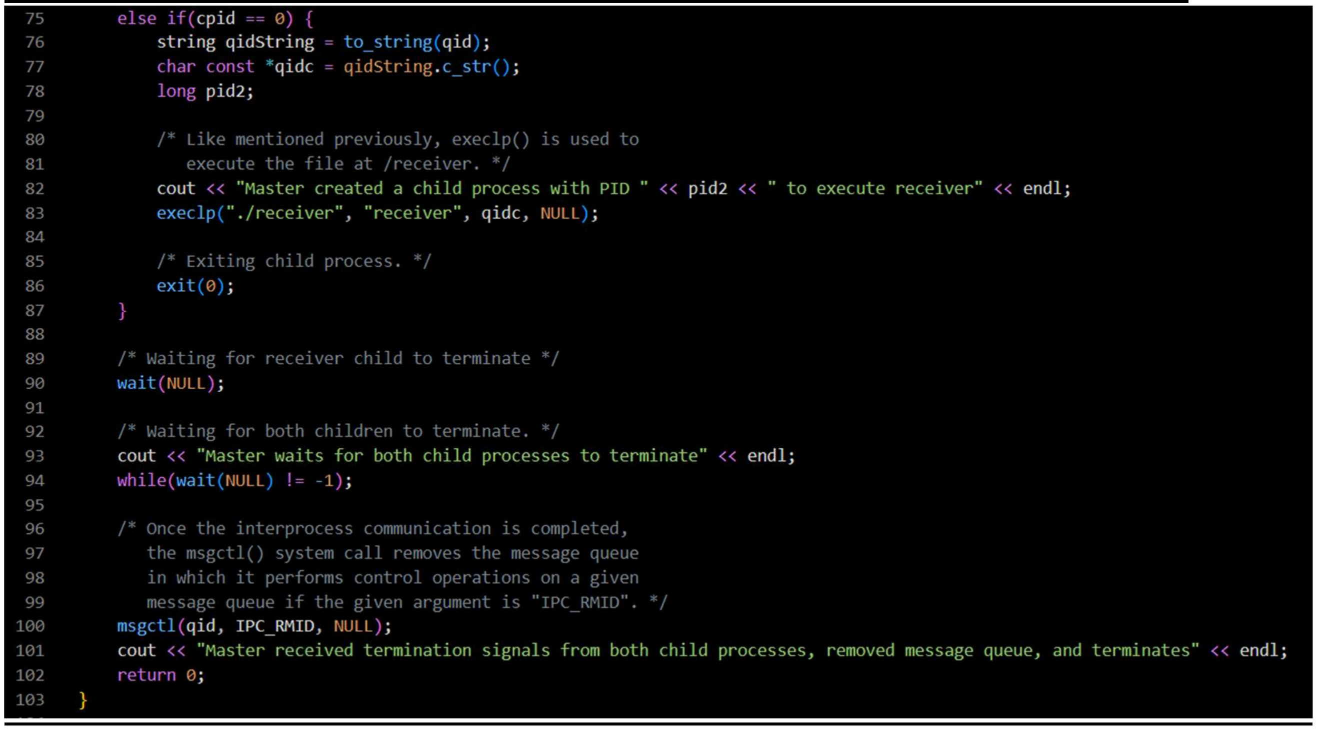
master.cc code:



```
/*
 *  Name: Dylan Dang
 *  ID: 026158052
 *  Course: CECS 326-01
 *  Professor: Shui Lam
 *  Program: Assignment 2 - Message Process
 *  File: master.cc
*/

#include <iostream>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

using namespace std;

int main() {

    /* The message queue is created via the msgget() system call
       in which the key_t type "IPC_PRIVATE" and the flag field
       "IPC_EXCL" return the identifier associated with the
       value of the key argument. */
    int qid = msgget(IPC_PRIVATE, IPC_EXCL|IPC_CREAT|0600);

    /* Master gets a PID and begins execution.
       Master also receives the message queue id */
    cout << "Master, PID " << getpid() << ", begins execution" << endl;
    cout << "Master acquired a message queue, id " << qid << endl;

    /* fork() system call is used to create the 1st child process - sender.
       A '0' means that the fork is successful. */
    pid_t cpid = fork();

```

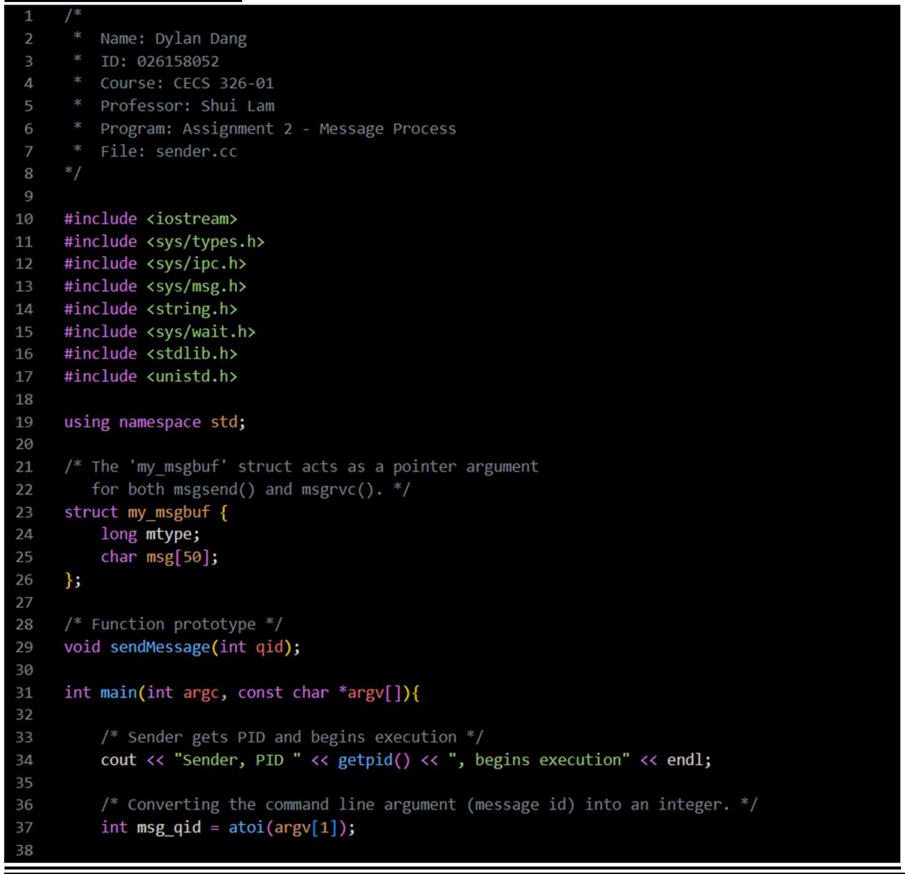


```cpp
    /* This block catches any failures during child process creation */
    if(cpid < 0) {
        cout << "Fork Failed! - child process not created." << endl;
        return 1;
    }

    /* After verifying no failures, sender is loaded in the 1st child
       and will generate the following messages. */
    else if(cpid == 0) {
        string qidString = to_string(qid);
        char const *qidc = qidString.c_str();
        long pid1;
        cout << "Master created a child process with PID " << pid1 << " to execute sender" << endl;

        /* The child process is executed via the execlp() system call
           which executed the file at the specified pathname
           which in this case is /sender. */
        execlp("./sender", "sender", qidc, NULL);
        sleep(1);

        /* Exiting child process. */
        exit(0);
    }

    /* Waiting for sender child to terminate. */
    wait(NULL);

    /* fork() is called again to create 2nd child process - receiver. */
    cpid = fork();

    /* Checks for failure in child process creation. */
    if(cpid < 0) {
        cout << "Fork Failed! - child process not created." << endl;
        return 1;
    }

```

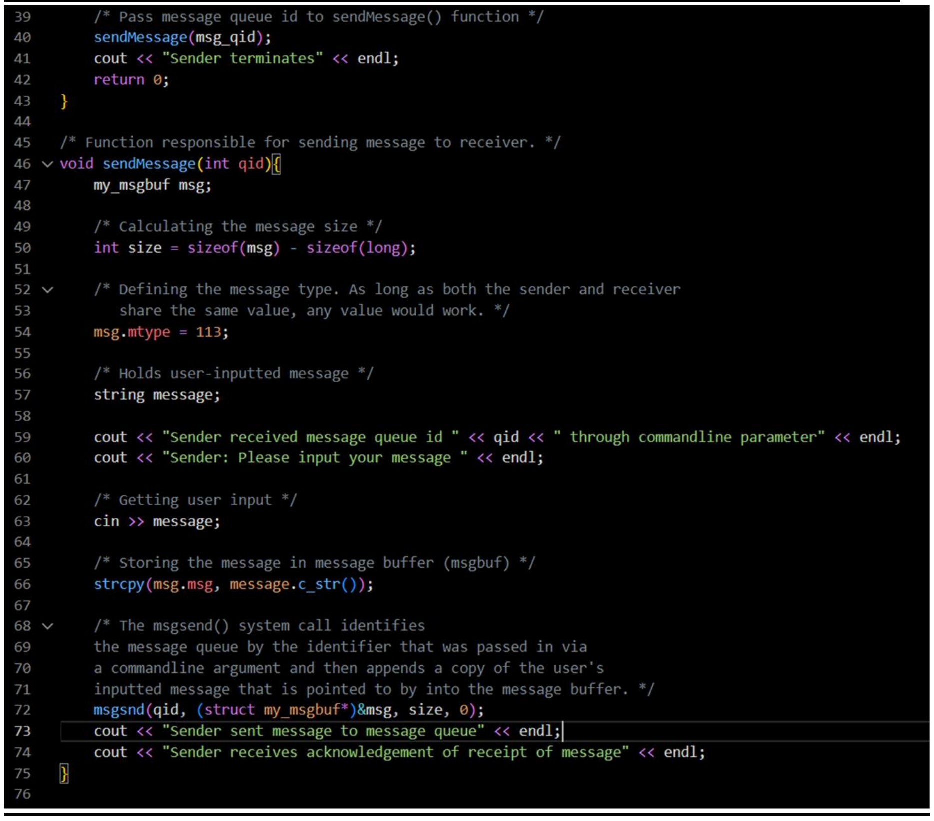


```cpp
    else if(cpid == 0) {
        string qidString = to_string(qid);
        char const *qidc = qidString.c_str();
        long pid2;

        /* Like mentioned previously, execlp() is used to
           execute the file at /receiver. */
        cout << "Master created a child process with PID " << pid2 << " to execute receiver" << endl;
        execlp("./receiver", "receiver", qidc, NULL);

        /* Exiting child process. */
        exit(0);
    }

    /* Waiting for receiver child to terminate */
    wait(NULL);

    /* Waiting for both children to terminate. */
    cout << "Master waits for both child processes to terminate" << endl;
    while(wait(NULL) != -1);

    /* Once the interprocess communication is completed,
       the msgctl() system call removes the message queue
       in which it performs control operations on a given
       message queue if the given argument is "IPC_RMID". */
    msgctl(qid, IPC_RMID, NULL);
    cout << "Master received termination signals from both child processes, removed message queue, and terminates" << endl;
    return 0;
}
```

sender.cc code:

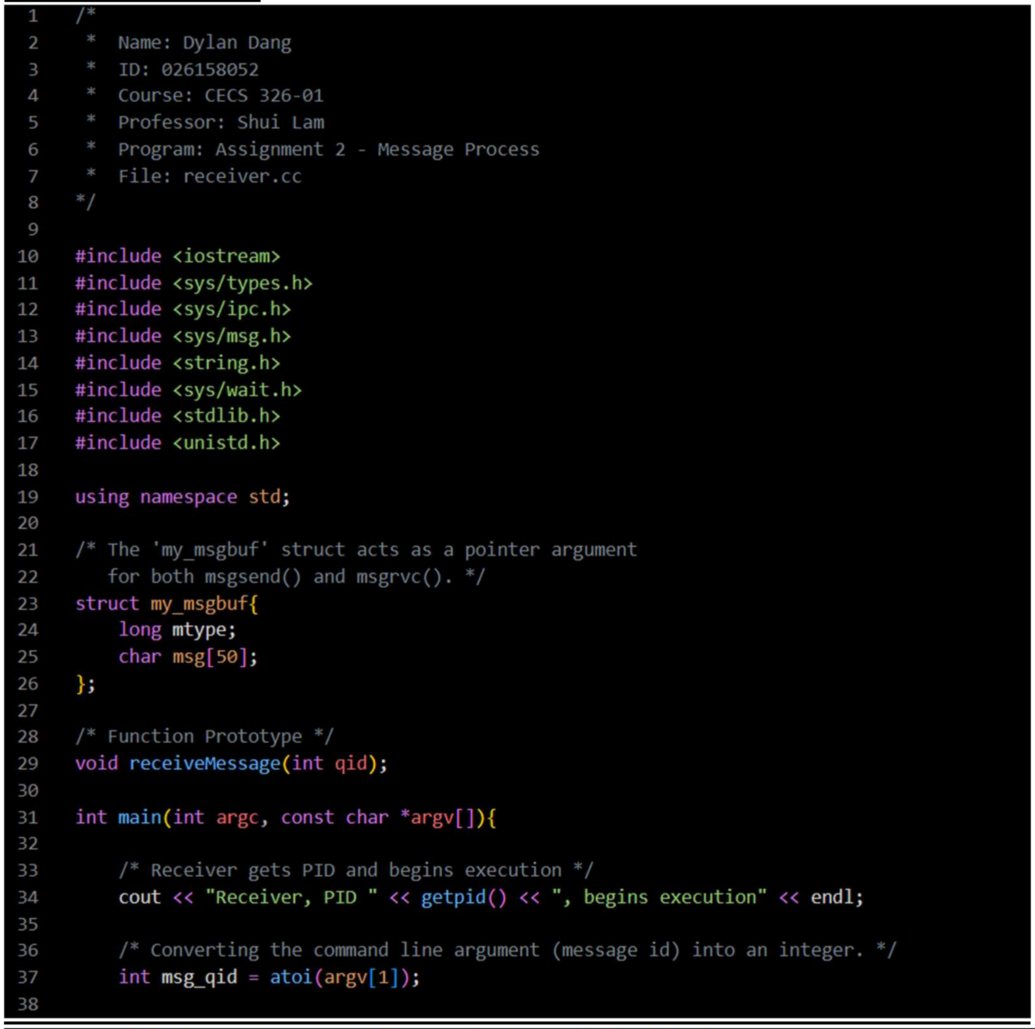


```cpp
/*
 *  Name: Dylan Dang
 *  ID: 026158052
 *  Course: CECS 326-01
 *  Professor: Shui Lam
 *  Program: Assignment 2 - Message Process
 *  File: sender.cc
*/

#include <iostream>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>

using namespace std;

/* The 'my_msgbuf' struct acts as a pointer argument
   for both msgsend() and msgrvc(). */
struct my_msgbuf {
    long mtype;
    char msg[50];
};

/* Function prototype */
void sendMessage(int qid);

int main(int argc, const char *argv[]){

    /* Sender gets PID and begins execution */
    cout << "Sender, PID " << getpid() << ", begins execution" << endl;

    /* Converting the command line argument (message id) into an integer. */
    int msg_qid = atoi(argv[1]);

```

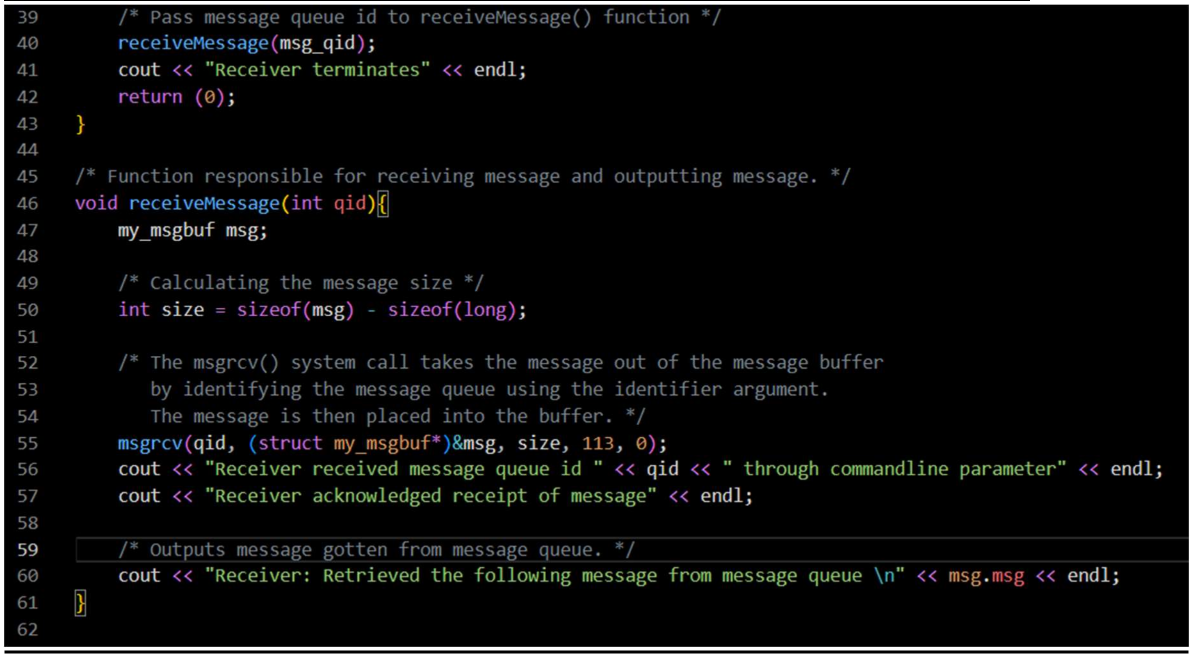


```cpp
    /* Pass message queue id to sendMessage() function */
    sendMessage(msg_qid);
    cout << "Sender terminates" << endl;
    return 0;
}

/* Function responsible for sending message to receiver. */
void sendMessage(int qid){
    my_msgbuf msg;

    /* Calculating the message size */
    int size = sizeof(msg) - sizeof(long);

    /* Defining the message type. As long as both the sender and receiver
       share the same value, any value would work. */
    msg.mtype = 113;

    /* Holds user-inputted message */
    string message;

    cout << "Sender received message queue id " << qid << " through commandline parameter" << endl;
    cout << "Sender: Please input your message " << endl;

    /* Getting user input */
    cin >> message;

    /* Storing the message in message buffer (msgbuf) */
    strcpy(msg.msg, message.c_str());

    /* The msgsend() system call identifies
    the message queue by the identifier that was passed in via
    a commandline argument and then appends a copy of the user's
    inputted message that is pointed to by into the message buffer. */
    msgsnd(qid, (struct my_msgbuf*)&msg, size, 0);
    cout << "Sender sent message to message queue" << endl;
    cout << "Sender receives acknowledgement of receipt of message" << endl;
}

```

receiver.cc code:



```
/*
 *  Name: Dylan Dang
 *  ID: 026158052
 *  Course: CECS 326-01
 *  Professor: Shui Lam
 *  Program: Assignment 2 - Message Process
 *  File: receiver.cc
*/

#include <iostream>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <unistd.h>

using namespace std;

/* The 'my_msgbuf' struct acts as a pointer argument
   for both msgsend() and msgrvc(). */
struct my_msgbuf{
    long mtype;
    char msg[50];
};

/* Function Prototype */
void receiveMessage(int qid);

int main(int argc, const char *argv[]){

    /* Receiver gets PID and begins execution */
    cout << "Receiver, PID " << getpid() << ", begins execution" << endl;

    /* Converting the command line argument (message id) into an integer. */
    int msg_qid = atoi(argv[1]);

```



```
    /* Pass message queue id to receiveMessage() function */
    receiveMessage(msg_qid);
    cout << "Receiver terminates" << endl;
    return (0);
}

/* Function responsible for receiving message and outputting message. */
void receiveMessage(int qid){
    my_msgbuf msg;

    /* Calculating the message size */
    int size = sizeof(msg) - sizeof(long);

    /* The msgrcv() system call takes the message out of the message buffer
       by identifying the message queue using the identifier argument.
       The message is then placed into the buffer. */
    msgrcv(qid, (struct my_msgbuf*)&msg, size, 113, 0);
    cout << "Receiver received message queue id " << qid << " through commandline parameter" << endl;
    cout << "Receiver acknowledged receipt of message" << endl;

    /* Outputs message gotten from message queue. */
    cout << "Receiver: Retrieved the following message from message queue \n" << msg.msg << endl;
}

```