```
// mainServer.c

/
*******************************************************************
***********
* Process Management Server: Main Server Process
* Author: Michael Hartung
* Date: 2/12/2015
* Description: This process is the main process that spawns child
server
* processes as well as sends signals to those servers to spawn child
processes.
* This processes also includes the ability to Display the Master/
Server/Process
* hierarchy.
*******************************************************************
***********/

#include "mainserver.h"

// Structure array for the servers (max of 20)
processStruct processes[MAX_SERVERS];

// Server counter variable
int numActive;

int main()
{
        // Register Handler for closing program
        signal(SIGINT, sigIntHandler);

        // Create the initial 2 servers
        createServer("2", "4", "FileServer");
        createServer("3", "5", "WebServer");

        // There are 2 currently active servers
        numActive = 2;
        char* inputString;
        while(1)
        {
                sleep(1);
                inputString = malloc(STRING_SIZE * sizeof(char));
                printf("\nEnter a command: ");
                fgets(inputString, STRING_SIZE, stdin);
                inputString = strtok(inputString, "\n");
                char* command = strtok(inputString, " ");
                if(command != NULL)
                {

                        //Create Server Command
```

```
                                if(!strcmp(command, "createServer"))
                                {
                                        char* minProcs = strtok(NULL, " ");
                                        char* maxProcs = strtok(NULL, " ");
                                        char* serverName = strtok(NULL, "
");
                                        createServer(minProcs, maxProcs,
serverName);
                                }

                                // Abort Server Command
                                else if(!strcmp(command, "abortServer"))
                                {
                                        char* serverName = strtok(NULL, "
");
                                        abortServer(serverName);
                                }

                                // Create Process Command
                                else if(!strcmp(command, "createProcess"))
                                {
                                        char *serverName = strtok(NULL, "
");
                                        int i;
                                        int serverIndex = -1;
                                        for(i = 0; i < numActive; i++)
                                        {
                                                if(!
strcmp(processes[i].serverName, serverName))
                                                {
                                                        serverIndex = i;
                                                }
                                        }
                                        if(serverIndex != -1)
                                        {

kill(processes[serverIndex].serverPid, SIGUSR1);
                                        }
                                        else
                                        {
                                                printf("Invalid Server!");
                                        }
                                }

                                // Abort Process Command
                                else if(!strcmp(command, "abortProcess"))
                                {
                                        char *serverName = strtok(NULL,
"\n");
                                        int i;
                                        int serverIndex = -1;
                                        for(i = 0; i < numActive; i++)
                                        {
                                                if(!
strcmp(processes[i].serverName, serverName))
                                                {
                                                        serverIndex = i;
                                                }
                                        }
                                        if(serverIndex >= 0)
                                        {

kill(processes[serverIndex].serverPid, SIGUSR2);
                                        }
                                        else
                                        {
                                                printf("Invalid Server!");
                                        }
                                }

                                // Display Status Command
                                else if(!strcmp(command, "displayStatus"))
                                {
                                        displayStatus();
                                }
                                else if(!strcmp(command, "exit"))
                                {
                                        sigIntHandler(0);
                                }
                                else {
                                        printf("Not a valid command!\n");
                                }
                        }
                        free(inputString);
                }
}


// Creates a server instance with Min/Max Processes under a name
(serverName)
void createServer(char* minProcs, char* maxProcs, char* serverName)
{
                if((processes[numActive].serverPid = fork()) < 0)
                {
                        printf("Fork failed!\n");
                }
                else if(!processes[numActive].serverPid)
                {
                        execl("server", minProcs, maxProcs, serverName,
NULL);
```

```
        }
        else
        {
                processes[numActive].serverName = malloc(STRING_SIZE
* sizeof(char));
                strcpy(processes[numActive].serverName, serverName);
                printf("Server %d created with name %s...\n",
processes[numActive].serverPid, processes[numActive].serverName);
        }
        numActive++;
}

// Aborts the server referenced as serverName
void abortServer(char* serverName)
{
        int abortIndex = 21;
        int i;
        for (i = 0; i < numActive; i++)
        {
                if(!strcmp(processes[i].serverName, serverName))
                {
                        abortIndex = i;
                }
        }
        if(abortIndex > 20)
        {
                printf("Server doesn't exist!\n");
        }
        else
        {
                int status = 0;
                int endID = 0;
                int i = 0;
                kill(processes[abortIndex].serverPid, SIGINT);
                while(!endID)
                        endID =
waitpid(processes[abortIndex].serverPid, &status, WNOHANG |
WUNTRACED);
                sleep(1);
                processes[abortIndex].serverPid = 0;
                free(processes[abortIndex].serverName);
                if(numActive > 0)
                {
                        for(i = abortIndex; i < numActive - 1; i++)
                        {
                                if(processes[i].serverPid == 0)
                                {
                                        processes[i].serverPid =
processes[i + 1].serverPid;
processes[abortIndex].serverName = malloc(STRING_SIZE * sizeof(char));

strcpy(processes[i].serverName, processes[i + 1].serverName);
                                        processes[i + 1].serverPid = 
0;
                                        free(processes[i + 
1].serverName);
                                }
                        }
                        numActive--;
                }
        }
}


// Handles the abortion of all servers and processes, then ends 
program
void sigIntHandler(int sigNum)
{
        int i = numActive - 1;
        while(i >= 0)
        {
                abortServer(processes[i].serverName);
                i--;
        }
        exit(0);
}

// Displays hierarchical view of Master/Server/Process structure
void displayStatus()
{
        int i;
        printf("--+= MainServer\n");
        for(i = 0; i < numActive; i++)
        {
                printf("  |\n  |----%s\n", processes[i].serverName);
        }
}
```