ELE4307

Real-Time Systems

Assignment 2

Student name: Kate Bowater

Student number: U1019160

Contents

[Introduction 3](#_Toc170216176)

[System Design 3](#_Toc170216177)

[Testing Results 3](#_Toc170216178)

[Appendix 3](#_Toc170216179)

# Introduction

This report documents the design of a C program which simulates a pick and place machine for assembly of Surface Mount Technology (SMT) based Printed Circuits Boards (PCBs). The program includes a controller and a simulator of which synchronization and inter-process communication between the two are the focus.

# System Design

Design choices, particularly with respect to the order of the process spawning, inter-process communication and synchronization.

# Testing Results

Test cases and results

The simulator and controller would not allow display to print to the terminal once running, and would give control back once the processes were terminated. The problem was found to be the memory mapped file; only when it is excluded from both the controller and simulator would the display then print everything that was sent through the pipes.

If the startup process terminates before the children, the display will print as it should when something comes in.

# Appendix

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Pick and Place Program

\* Startup and Monitoring main file

\* By: Kate Bowater

\* Student #: U1019160

\*

\* This file creates forks and pipes for communication

\* between the Display, Simulator, and Controller

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <string.h>

#define NUMBER\_OF\_CHILDREN 1

#define CHILD 0

#define FORK\_FAILED -1

#define READ 0

#define WRITE 1

//#define READ\_BLOCK\_SIZE 9

int pipe\_Startup\_to\_Display[2]; //[0] for read, [1] for write

int pipe\_Simulator\_to\_Display[2];

int main()

{

//pipe details

//char readBuffer[READ\_BLOCK\_SIZE+1];

//ssize\_t bytesRead;

char \*strFromStartup = "Startup: Here is a message from Startup to Display\n";

char pipeStartupToDisplayReadFdStr[10];

//end pipe details

int DisplayStatus; //for parent to monitor the status of the display child

// int SimulatorStatus; //for parent to monitor the status of the simulator child

//set up pipe before fork

if (pipe(pipe\_Startup\_to\_Display) < 0)

{

perror("Pipe creation failed");

exit(5);

}

// string creation for read end of pipe, to allow display to read from startup

sprintf(pipeStartupToDisplayReadFdStr, "%d", pipe\_Startup\_to\_Display[READ]);

for (int count = 0; count < NUMBER\_OF\_CHILDREN; count++)

{

pid\_t return\_pid = fork();

//using switch to organise the child and parent processes

switch (return\_pid)

{

case CHILD:

if (count == 0) //first is the display process

{

printf("Startup: Display process created with PID %d. Going to overlay.\n", getpid());

close(pipe\_Startup\_to\_Display[WRITE]); //display will only be reading through the pipes

close(pipe\_Simulator\_to\_Display[WRITE]);

//char countString[2];

//sprintf(countString, "%i", count + 1);

execl("..\\Assgn2\_2024\_Display\\bin\\Release\\Assgn2\_2024\_Display", "Assgn2\_2024\_Display", pipeStartupToDisplayReadFdStr, (char \*) NULL);

perror("Display overlay failed: ");

exit(5);

}

if(count == 1) // second is the simulator

{

close(pipe\_Simulator\_to\_Display[READ]); //Simulator will only write to pipe

// strFromStartup = "Startup: Simulator process created with PID ";

// write(pipe\_Startup\_to\_Display[WRITE], strFromStartup, strlen(strFromStartup));

// sprintf(strFromStartup, "%i. Going to overlay ", getpid());

// write(pipe\_Startup\_to\_Display[WRITE], strFromStartup, strlen(strFromStartup));

//strFromStartup = "Startup: Simulator process created with PID %d. Going to overlay.\n", getpid();

//write(pipe\_Startup\_to\_Display[WRITE], strFromStartup, strlen(strFromStartup));

//printf("Startup: Simulator process created with PID %d. Going to overlay.\n", getpid());

execl("..\\Assgn2\_2024\_Simulator\\bin\\Release\\Assgn2\_2024\_Simulator", "Assgn2\_2024\_Simulator", (char \*) NULL);

perror("Simulator overlay failed: ");

exit(5);

}

if(count == 2) // final child is the Controller

{

printf("Startup: Controller process created with PID %d. Going to overlay.\n", getpid());

execl("..\\Assgn2\_2024\_Controller\\bin\\Release\\Assgn2\_2024\_Controller", "Assgn2\_2024\_Controller", (char \*) NULL);

perror("Controller overlay failed: ");

exit(5);

}

case FORK\_FAILED:

perror("Fork failed: ");

exit(5);

// the Startup process continues here as the parent

default:

if(count == 0)

{

close(pipe\_Startup\_to\_Display[READ]); //parent will only write to pipe, so close read end

printf("Startup: Writing strFromStartup to pipe\n");

write(pipe\_Startup\_to\_Display[WRITE], strFromStartup, strlen(strFromStartup));

// bytesRead = read(pipe\_Startup\_to\_Display[READ], readBuffer, READ\_BLOCK\_SIZE); //startup will stop here until there is a message

// printf("Startup process reading message:\n");

// while (bytesRead > 0)

// {

// readBuffer[bytesRead] = '\0';

// printf("bytes:%li\n'%s'\n", bytesRead, readBuffer);

// bytesRead = read(pipe\_Startup\_to\_Display[READ], readBuffer, READ\_BLOCK\_SIZE);

// }

}

if(count == 1)

{

strFromStartup = "Startup: Simulator process created with PID ";

write(pipe\_Startup\_to\_Display[WRITE], strFromStartup, strlen(strFromStartup));

sprintf(strFromStartup, "%i. Going to overlay ", getpid());

write(pipe\_Startup\_to\_Display[WRITE], strFromStartup, strlen(strFromStartup));

//print next message to Display

//strFromStartup = "Startup process has created Simulator child.\n";

//write(pipe\_Startup\_to\_Display[WRITE], strFromStartup, strlen(strFromStartup));

}

if(count == 2)

{

//print next message to Display

//strFromStartup = "Startup process has created Controller child.\n";

//write(pipe\_Startup\_to\_Display[WRITE], strFromStartup, strlen(strFromStartup));

}

} // end switch

}//end for loop

close(pipe\_Simulator\_to\_Display[READ]); //Startup does not need communication to Simulator

close(pipe\_Simulator\_to\_Display[WRITE]); //Startup does not need communication to Simulator

strFromStartup = "Startup: Process spawning completed\n";

write(pipe\_Startup\_to\_Display[WRITE], strFromStartup, strlen(strFromStartup));

//close(pipe\_Startup\_to\_Display[WRITE]);

//Startup will wait for the children to terminate

// for (int count = 0; count < NUMBER\_OF\_CHILDREN; count++)

// {

// int status;

//// if (count == NUMBER\_OF\_CHILDREN)

//// {

//// strFromStartup = "Startup closing pipe to Display\n";

//// write(pipe\_Startup\_to\_Display[WRITE], strFromStartup, strlen(strFromStartup));

//// close(pipe\_Startup\_to\_Display[WRITE]); // finished writing to the pipe

//// pid\_t return\_pid = wait(&status);

//// printf("Startup: Display process with PID %i terminated ", return\_pid);

//// printf("with status %i\n", status>>8);

//// }

//// else

//// {

// pid\_t return\_pid = wait(&status);

//

// sprintf(strFromStartup, "Startup: Process with PID %i ", return\_pid);

// write(pipe\_Startup\_to\_Display[WRITE], strFromStartup, strlen(strFromStartup));

// sprintf(strFromStartup, "terminated with status %i\n", status>>8);

// write(pipe\_Startup\_to\_Display[WRITE], strFromStartup, strlen(strFromStartup));

// //printf("Startup: Child process with PID %i terminated ", return\_pid);

// //printf("with status %i\n", status>>8);

//// }

//

//

// }

//Once all children have terminated

strFromStartup = "Startup: Waiting for Display process to terminate\n";

write(pipe\_Startup\_to\_Display[WRITE], strFromStartup, strlen(strFromStartup));

close(pipe\_Startup\_to\_Display[WRITE]);

pid\_t DisplayPID = wait(&DisplayStatus); // waiting for the display process to exit

printf("Startup: Display with PID %d terminated with status code %d.\n", DisplayPID, DisplayStatus>>8);

exit(0);

}//end main