Swapnil Acharya

StarId: ea4963aw Due: 02/05/2020

Purpose

To increase your understanding of system calls

CSCI 610  
 Advanced concepts in operating systems

Project 1

# @author Swapnil Acharya

# @since 01/23/2020

# @file Makefile

all**:** compute controller

compute**:** P1.h P1\_compute.c

gcc -o compute P1.h P1\_compute.c

controller**:** P1.h P1\_controller.c

gcc -o controller P1.h P1\_controller.c

clean**:**

rm -f compute

rm -f controller

/\*\*

\* **@author** Swapnil Acharya

\* **@since** 01/23/2020

\* **@file** P1.h

\*/

#ifndef \_P1\_H\_

#define \_P1\_H\_

//number of child processes

#define MAXCP 2

//define buffer lengths for commands between parent and child processes

#define CMD\_LEN 250

//commands used to communicate between controller

//and compute

#define READY "READY"

#define COMPUTE "COMPUTE"

#define DONE "DONE"

#define PRINT "PRINT"

//define process id for two child processes

#define COMPUTE\_ID0 0

#define COMPUTE\_ID1 1

//define increment value

#define INC 1000

#endif

/\*\*

\* **@author** Swapnil Acharya

\* **@since** 01/23/2020

\* **@file** P1\_controller.c

\*/

//include custom files

#include "P1.h"

//include standard libary

#include <stdio.h>

#include <stdlib.h>

//include required linux headers

#include <unistd.h>

#include <errno.h>

#include <sys/types.h>

//include other headers

#include <string.h>

#include <time.h>

#include <stdbool.h>

**typedef** struct \_compTime**{**

time\_t \_startTime**;**

time\_t \_endTime**;**

double \_totalTime**;**

**}** ComputationTime**;**

int main**(**int \_argc**,** char **\***\_argv**[],** char **\***\_envp**[]){**

//container to hold child process computaion time

ComputationTime \_cTime**[**MAXCP**];**

//intialize variables

int i **=** 0**;**

**for(**i **=** 0**;** i **<** MAXCP**;** i**++){**

\_cTime**[**i**].**\_startTime **=** 0**;**

\_cTime**[**i**].**\_endTime **=** 0**;**

\_cTime**[**i**].**\_totalTime **=** 0**;**

**}**

//define pipes for communication between controller and players (Compute Processes)

int \_controllerWriteComputeRead**[**MAXCP**][**2**];**

int \_computeWriteControllerRead**[**MAXCP**][**2**];**

//define messasge buffers for communication between parent (controller) and child (compute) processes

char \_readStream**[**MAXCP**][**CMD\_LEN**],** \_writeStream**[**MAXCP**][**CMD\_LEN**];**

//define buffers for read and write

char \_readBuffer**[**CMD\_LEN**];**

char \_writeBuffer**[**CMD\_LEN**];**

//create pipes for controller to write and computes to read

int \_err**;**

**for(**i **=** 0**;** i **<** MAXCP**;** i**++){**

\_err **=** pipe**(**\_controllerWriteComputeRead**[**i**]);**

**if(**\_err **==** 1**){**

printf**(**"Controller: Error creating \_controllerWriteComputeRead[%i], errno =%d\n\n"**,**errno**,**i**);**

exit**(**1**);**

**}**

**}**

//create pipes for computes to write and contoller to read

**for(**i **=** 0**;** i **<** MAXCP**;** i**++){**

\_err **=** pipe**(**\_computeWriteControllerRead**[**i**]);**

**if(**\_err **==** 1**){**

printf**(**"Controller: Error creating \_computeWriteControllerRead[%i], errno =%d\n\n"**,**errno**,**i**);**

exit**(**2**);**

**}**

**}**

//create child processes

pid\_t \_processId**[**MAXCP**];**

**for(**i**=**0**;** i **<** MAXCP**;** i**++){**

//storing ids in an array,

//In parent process, the ids of child processes are stored in this array

//If this array is later accesed in parent process, the id of child processes can be accessed

\_processId**[**i**]** **=** fork**();**

**if(**\_processId**[**i**]** **<** 0**){**

printf**(**"Error creating child Process, Process Id =%d, Errno %d\n\n"**,**getpid**(),**errno**);**

exit**(**3**);**

**}**

**else** **if(**\_processId**[**i**]** **==** 0**){** //child process

//close unused pipe ends for communication with controller

close**(**\_computeWriteControllerRead**[**i**][**0**]);** //close read end of pipe used for child process to send command to parent process

close**(**\_controllerWriteComputeRead**[**i**][**1**]);** //close write end of pipe used for parent process to send command to child process

//convert file/stream descriptor from integer to string

sprintf**(**\_readStream**[**i**],**"%d"**,** \_controllerWriteComputeRead**[**i**][**0**]);** //read stream

sprintf**(**\_writeStream**[**i**],**"%d"**,** \_computeWriteControllerRead**[**i**][**1**]);**//write stream

//execute compute program in child process

\_err **=** execl**(**"./compute"**,**\_readStream**[**i**],** \_writeStream**[**i**],** **(**char **\*)NULL);**

**if(**\_err **==** **-**1**){**

printf**(**"Error excuting Execl, Child PID %d, errno%d\n\n"**,**getpid**(),**errno**);**

exit**(**4**);**

**}**

**}**

**}**

//parent process

//close unsused pipes end for communication with compute processes

**for(**i **=**0**;** i **<** MAXCP**;** i**++){**

close**(**\_controllerWriteComputeRead**[**i**][**0**]);** //close read end of pipe used for parent process to send command to child processes

close**(**\_computeWriteControllerRead**[**i**][**1**]);**//close write end of pipe used for child process to send command to parent process

**}**

//parent process

//wait for READY command from all child(compute processes)

**for(**i **=** 0**;** i **<** MAXCP**;** i**++){**

//read

\_err **=** read**(**\_computeWriteControllerRead**[**i**][**0**],**\_readBuffer**,**CMD\_LEN**);**

**if(**\_err **==** **-**1**){**

printf**(**"Error while reading from \_computeWriteControllerRead[%d][0] , PID:%d, errno%d\n\n"**,**i**,**getpid**(),**errno**);**

exit**(**5**);**

**}**

//validate if read command is READY

**if(**strncmp**(**\_readBuffer**,**READY**,**strlen**(**READY**))** **==** 0**){**

printf**(**"controller: %s received from child process(%d).\n"**,**READY**,**\_processId**[**i**]);**

**}**

**else{**

printf**(**"Response from Compute %i mismatch. \n"**,**i**);**

exit**(**6**);**

**}**

//clear readbuffer

memset**(**\_readBuffer**,**0**,**CMD\_LEN**);**

**}**

//parent process

//send COMPUTE To Child processes and

//start computation time for processes

**for(**i **=** 0**;** i **<** MAXCP**;** i**++){**

//start computaion time for all compute processes

time**(&(**\_cTime**[**i**].**\_startTime**));**

//send COMPUTE to child process

printf**(**"controller: Sending %s to child process(%d).\n"**,**COMPUTE**,**\_processId**[**i**]);**

//PUT ID = 0 to odd and 1 to even loop index and add COMPUTE command into buffer to be sent to child processes

**if(**i **%** 2 **==** 0**){**

sprintf**(**\_writeBuffer**,**"%s\_%i\_%i"**,**COMPUTE**,**COMPUTE\_ID0**,**INC**);**

**}**

**else{**

sprintf**(**\_writeBuffer**,**"%s\_%i\_%i"**,**COMPUTE**,**COMPUTE\_ID1**,**INC**);**

**}**

//send COMPUTE command and ID to child processes

\_err **=** write**(**\_controllerWriteComputeRead**[**i**][**1**],**\_writeBuffer**,**strlen**(**\_writeBuffer**)+**1**);**

**if(**\_err **==** **-**1**){**

printf**(**"Error while writting to \_controllerWriteComputeRead[%i][1], PID:%d, errno:%d\n\n"**,**i**,**getpid**(),** errno**);**

exit**(**7**);**

**}**

**}**

//parent process

//wait for DONE command from all child(compute processes)

**for(**i **=** 0**;** i **<** MAXCP**;** i**++){**

//read

\_err **=** read**(**\_computeWriteControllerRead**[**i**][**0**],**\_readBuffer**,**CMD\_LEN**);**

**if(**\_err **==** **-**1**){**

printf**(**"Error while reading from \_computeWriteControllerRead[%d][0] , PID:%d, errno%d\n\n"**,**i**,**getpid**(),**errno**);**

exit**(**8**);**

**}**

//validate if read command is DONE

**if(**strncmp**(**\_readBuffer**,**DONE**,**strlen**(**DONE**))** **==** 0**){**

printf**(**"controller: %s received from child process(%d).\n"**,**DONE**,**\_processId**[**i**]);**

**}**

**else{**

printf**(**"Response from Compute %i mismatch. \n"**,**i**);**

exit**(**9**);**

**}**

//Measure end time taken to recevived DONE commands from child processes

time**(&(**\_cTime**[**i**].**\_endTime**));**

//clear readbuffer

memset**(**\_readBuffer**,**0**,**CMD\_LEN**);**

**}**

//parent process

//calculate total computaion time taken for all child processes

**for(**i **=** 0**;** i **<** MAXCP**;** i**++){**

\_cTime**[**i**].**\_totalTime **=** difftime**(**\_cTime**[**i**].**\_endTime**,** \_cTime**[**i**].**\_startTime**);**

printf**(**"controller: Total Computaion Time for child process(%d): %f seconds\n"**,**\_processId**[**i**],**\_cTime**[**i**].**\_totalTime**);**

**}**

//parent process

//send PRINT To Child processes and

//start computation time for processes

**for(**i **=** 0**;** i **<** MAXCP**;** i**++){**

//send PRINT to child process

printf**(**"controller: Sending %s to child process(%d).\n"**,**PRINT**,**\_processId**[**i**]);**

//send PRINT command and ID to child processes

sprintf**(**\_writeBuffer**,**"%s"**,**PRINT**);**

\_err **=** write**(**\_controllerWriteComputeRead**[**i**][**1**],**\_writeBuffer**,**strlen**(**\_writeBuffer**)+**1**);**

**if(**\_err **==** **-**1**){**

printf**(**"Error while writting PRINT to \_controllerWriteComputeRead[%i][1], PID:%d, errno:%d\n\n"**,**i**,**getpid**(),** errno**);**

exit**(**10**);**

**}**

**}**

//wait for child processes to exit

**for(**i **=** 0**;** i **<** MAXCP**;** i**++){**

int \_status **=** 0**;**

waitpid**(**\_processId**[**i**],&**\_status**,**0**);**

**if(**WIFEXITED**(**\_status**)){**

printf**(**"controller: child(%d) terminated with exit status: %d\n"**,**\_processId**[**i**],**WEXITSTATUS**(**\_status**));**

**}**

**}**

printf**(**"controller: Complete.\n"**);**

exit**(**11**);**

**}**

/\*\*

\* **@author** Swapnil Acharya

\* **@since** 01/23/2020

\* **@file** P1\_compute.c

\*/

//include custom files

#include "P1.h"

//include standard library

#include <stdio.h>

#include <stdlib.h>

//include required linux headers

#include <unistd.h>

#include <errno.h>

#include <sys/types.h>

//include other headers

#include <string.h>

int getRelationIsTrueCount**(**int \_start**,** int \_end**){**

//c => [id\*INC, (id+1)\*INC)

// a, b are integers in the interval (0,c)

//intialie variables

int a **=** 0**,** b **=**0**,** c **=** 0**;**

int \_count **=** 0**;** //variable to hold the number of count when c^2 = a^2 + b^2 is true

**for(**c **=** \_start**;** c **<** \_end**;** c**++** **){**

**for(**a **=** 1**;** a **<** c**;** a**++){**

**for(**b **=** 1**;** b **<** c**;** b**++){**

int \_cSquared **=** c **\*** c**;**

int \_abSquared **=** **(**a **\*** a**)** **+** **(**b **\*** b**);**

**if(**\_cSquared **==** \_abSquared**){**

\_count **=** \_count **+** 1**;**

**}**

**}**

**}**

**}**

**return** \_count**;**

**}**

int main**(**int \_argc**,** char **\***\_argv**[],** char **\***\_envp**[]){**

char \_readBuffer**[**CMD\_LEN**];**

char \_writeBuffer**[**CMD\_LEN**];**

//get stream pipes

int \_readPipe **=** atoi**(**\_argv**[**0**]);**

int \_writePipe **=** atoi**(**\_argv**[**1**]);**

//sending READY to parent process

printf**(**"compute(%d): Sending %s to parent process. \n"**,**getpid**(),** READY**);**

int \_error **=** write**(**\_writePipe**,**READY**,**strlen**(**READY**)+**1**);**

**if(**\_error **==** **-**1**){**

printf**(**"Error writting to pipe, CID: %d, errno:%d\n\n"**,**getpid**(),**errno**);**

exit**(**1**);**

**}**

//define computaion parameters

int \_id **=** 0**;** //variable to hold id value

int \_INC **=** 0**;** //variable too hold increment value

int \_startInterval **=** 0**;**

int \_endInterval **=** 0**;**

int \_count **=** 0**;**

//wait for compute from parent process

\_error **=** read**(**\_readPipe**,**\_readBuffer**,**CMD\_LEN**);**

**if(**\_error **==** **-** 1**){**

printf**(**"Error reading from pipe, CID:%d, errno:%d\n\n"**,**getpid**(),**errno**);**

exit**(**2**);**

**}**

**else{**

**if(**strncmp**(**\_readBuffer**,**COMPUTE**,**strlen**(**COMPUTE**))** **==** 0**){**

printf**(**"compute(%d): %s recevied.\n"**,**getpid**(),**COMPUTE**);**

//extract data from buffer

char **\*** \_delimiterFinder**;**

\_delimiterFinder **=** strtok**(**\_readBuffer**,**"\_"**);** //extract compute

\_delimiterFinder **=** strtok**(NULL,**"\_"**);** //extract id

\_id **=** atoi**(**\_delimiterFinder**);** //convert id from string to integer

\_delimiterFinder **=** strtok**(NULL,**"\_"**);**

\_INC **=** atoi**(**\_delimiterFinder**);** //convert INC from string to integer

**}**

**else{**

printf**(**"Command Mismatch: %s instead of %s, CID: %d, errno:%d\n\n"**,**\_readBuffer**,**COMPUTE**,**getpid**(),**errno**);**

exit**(**3**);**

**}**

**}**

//start computation

//c => [id\*INC, (id+1)\*INC)

\_startInterval **=** \_id **\*** \_INC**;**

\_endInterval **=** **(**\_id **+** 1**)** **\*** \_INC**;**

\_count **=** getRelationIsTrueCount**(**\_startInterval**,**\_endInterval**);**

//send DONE TO Controller

printf**(**"compute(%d): Sending %s to parent process. \n"**,**getpid**(),** DONE**);**

\_error **=** write**(**\_writePipe**,**DONE**,**strlen**(**DONE**)+**1**);**

**if(**\_error **==** **-**1**){**

printf**(**"Error writting to pipe, CID: %d, errno:%d\n\n"**,**getpid**(),**errno**);**

exit**(**4**);**

**}**

//wait for print command from parent process

\_error **=** read**(**\_readPipe**,**\_readBuffer**,**CMD\_LEN**);**

**if(**\_error **==** **-** 1**){**

printf**(**"Error reading from pipe, CID:%d, errno:%d\n\n"**,**getpid**(),**errno**);**

exit**(**5**);**

**}**

**else{**

**if(**strncmp**(**\_readBuffer**,**PRINT**,**strlen**(**PRINT**))** **==** 0**){**

printf**(**"compute(%d): %s recevied.\n"**,**getpid**(),**PRINT**);**

**}**

**else{**

printf**(**"Command Mismatch: %s instead of %s, CID: %d, errno:%d\n\n"**,**\_readBuffer**,**COMPUTE**,**getpid**(),**errno**);**

exit**(**6**);**

**}**

**}**

//print compute parameters and resuls

printf**(**"compute(%d): For interval [%d,%d): count = %d \n"**,**getpid**(),**\_startInterval**,**\_endInterval**,**\_count**);**

//Indicate Exit

printf**(**"compute(%d): Terminating Normally.\n"**,**getpid**());**

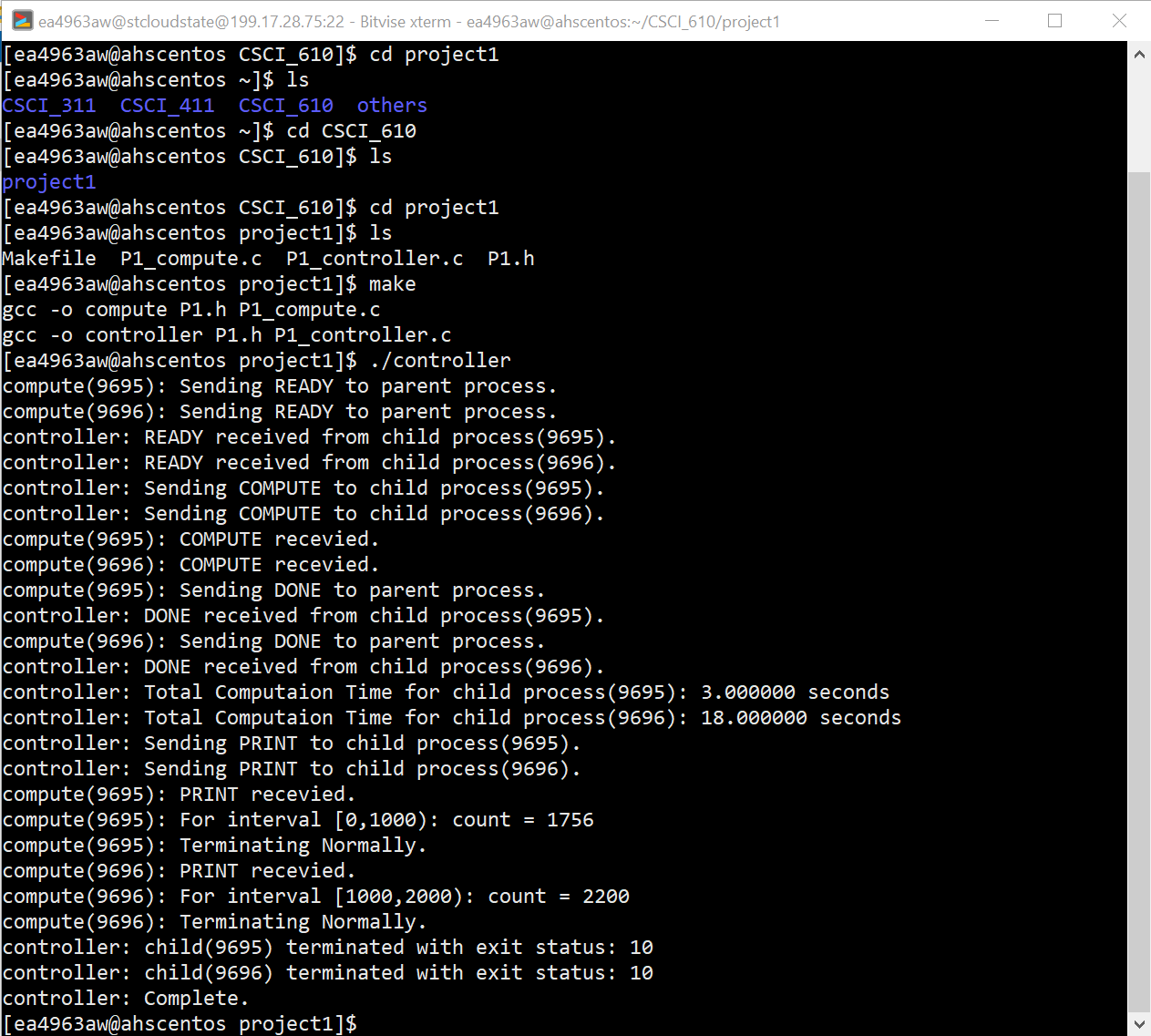
exit**(**10**);**

**return** 0**;**

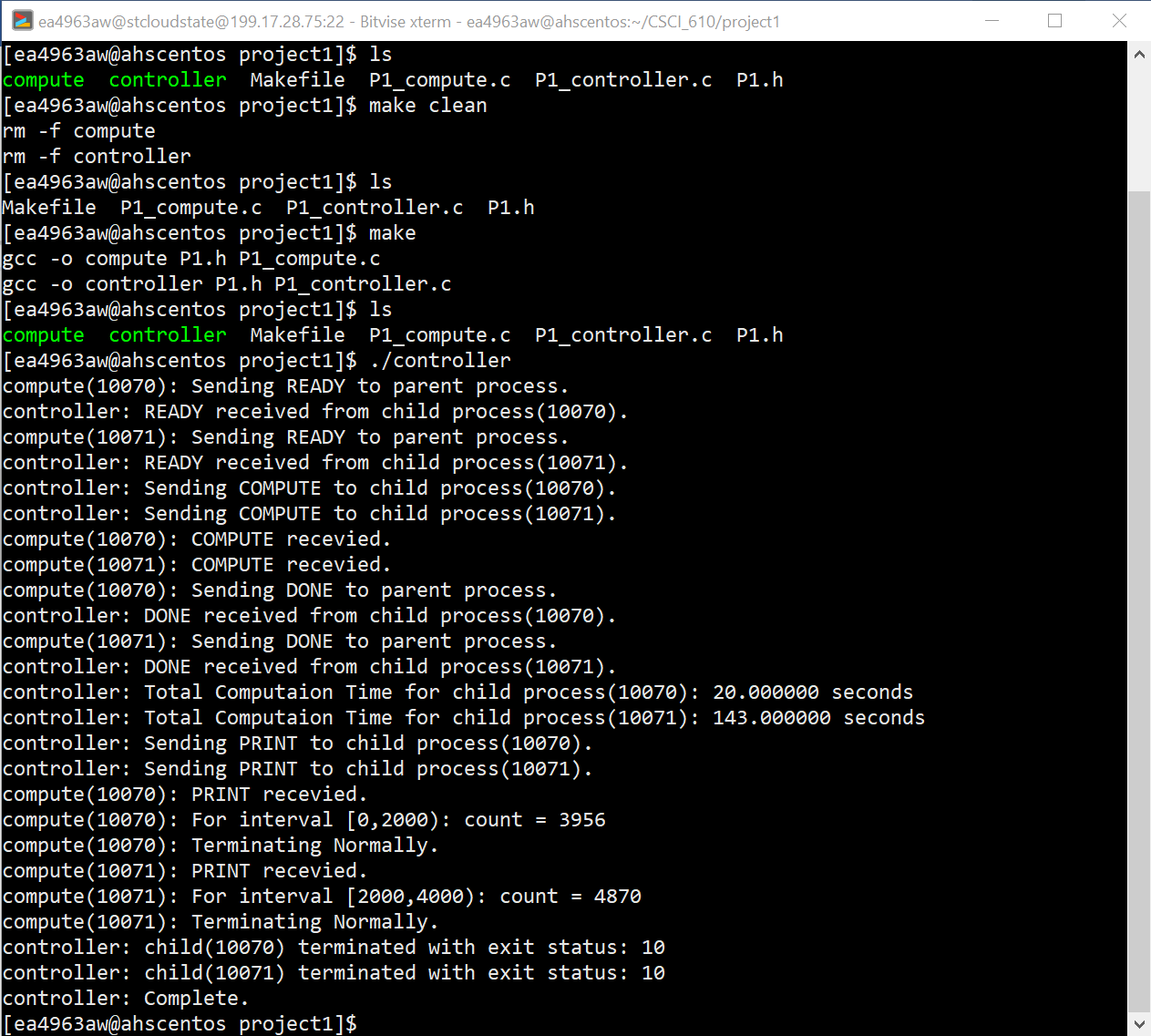
**}**

**OUTPUTS**

**MAXCP = 2, INC = 1000**

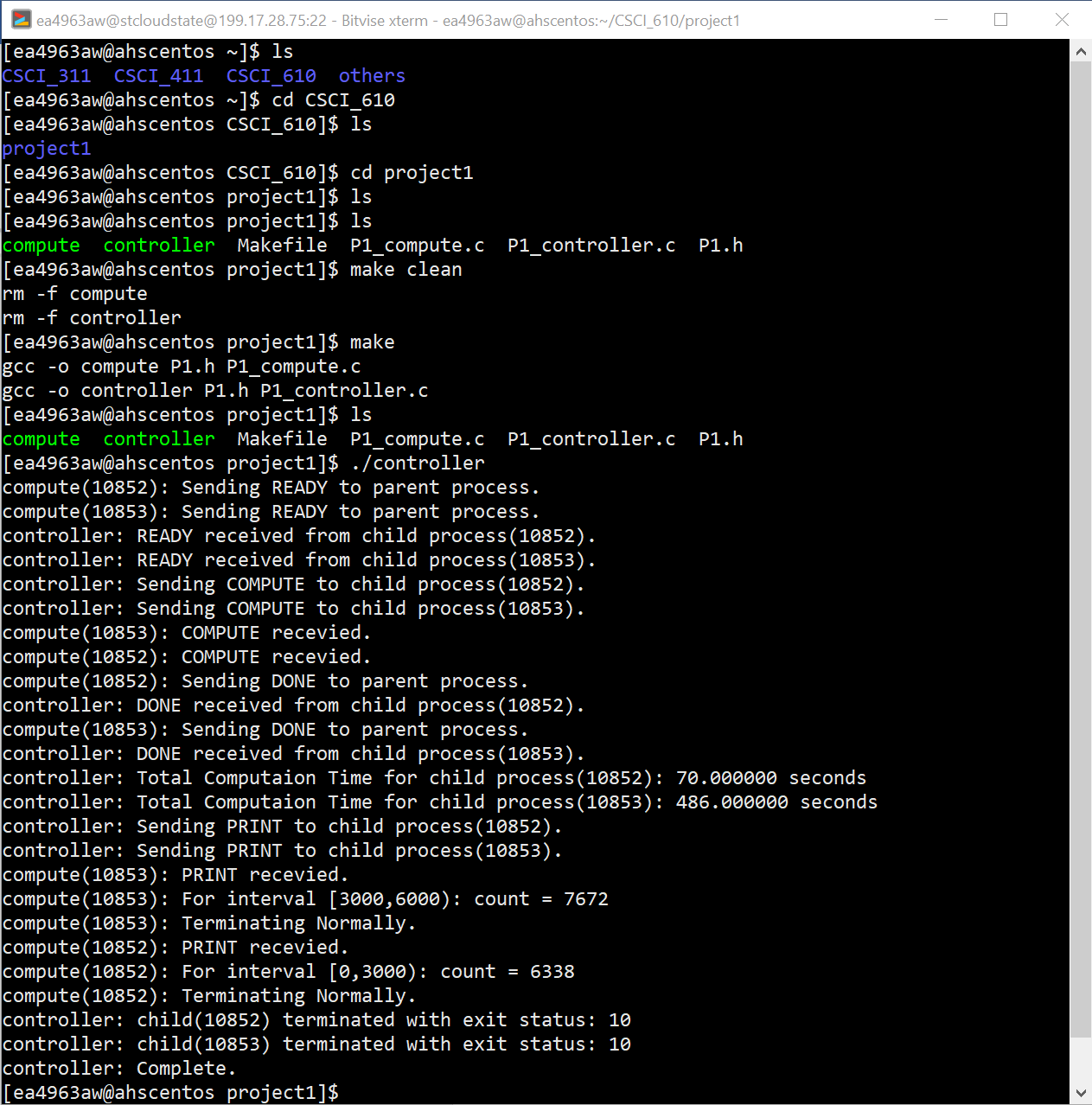
****

**MAXCP = 2, INC = 2000**

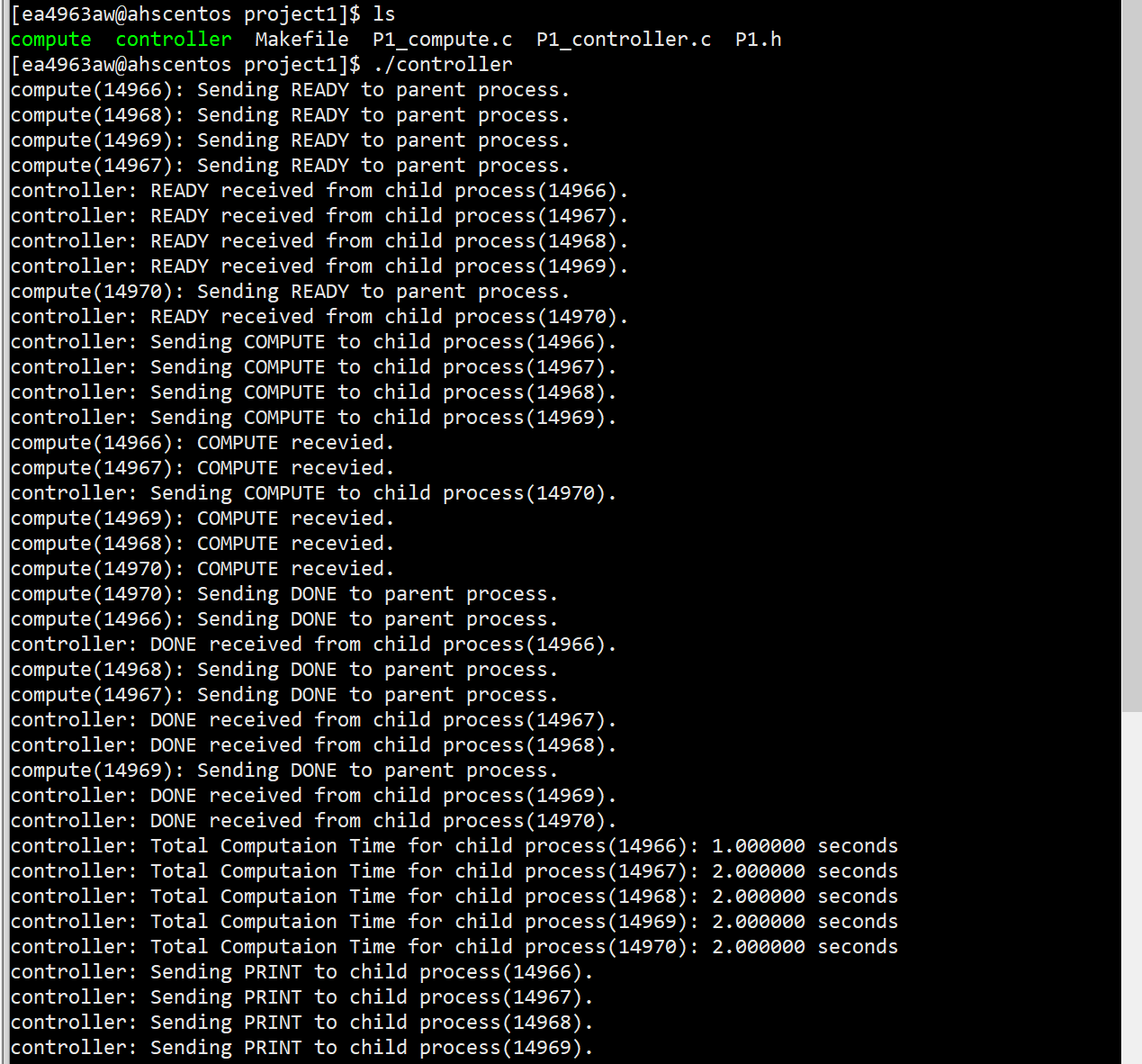
****

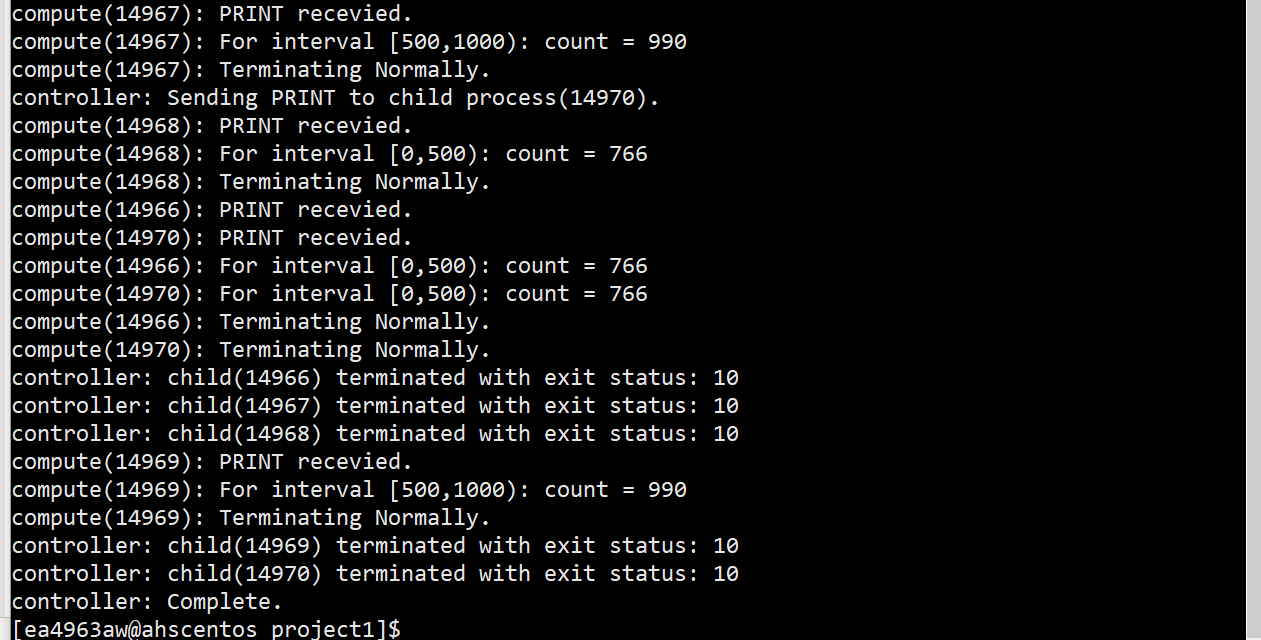
**MAXCP = 2, INC = 3000**



****

**MAXCP = 5, INC = 500**

****

****