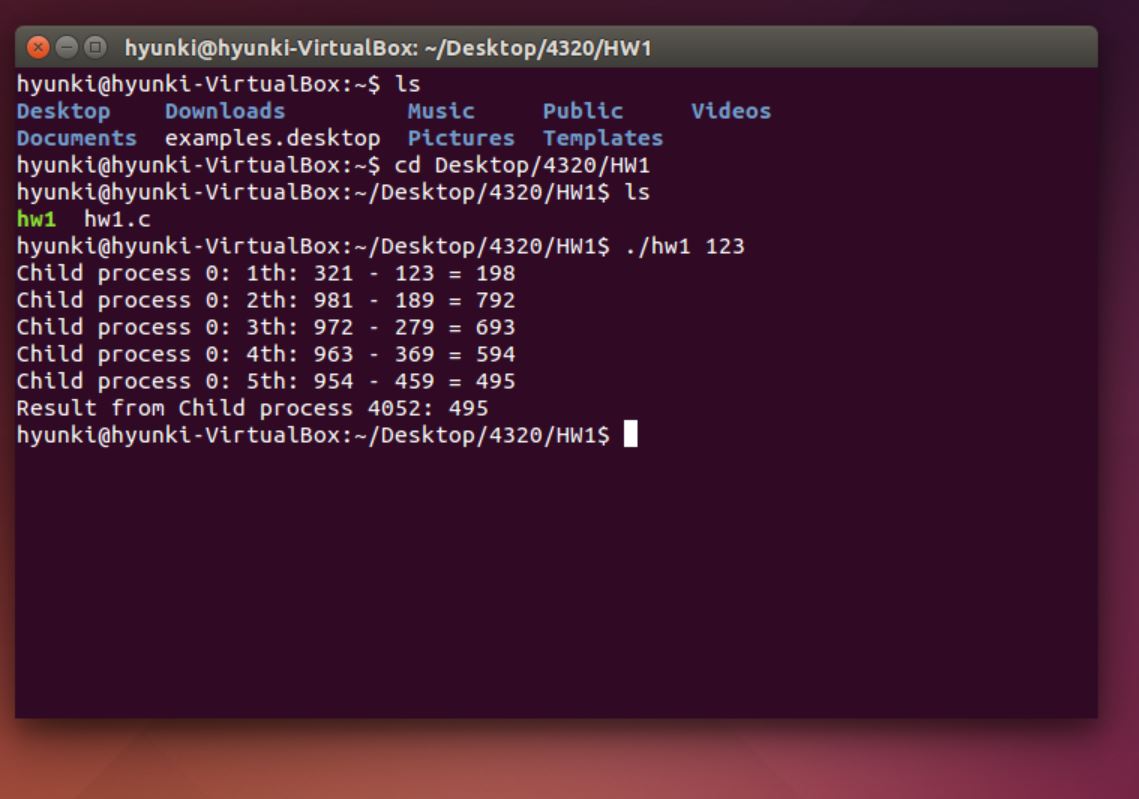
HomeWork1

CSC4320

#002-34-4677

Hyunki Lee

1. Output Screenshot



1. Copy of my C source code

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <string.h>

#include <stdlib.h>

#define BUFFER\_SIZE 10

#define READ\_END 0

#define WRITE\_END 1

int main(int argc, char \*argv[])

{

char buff[BUFFER\_SIZE]; //create buffer

int n;

int fd[2]; //need 2 factors in order to read and write pipe

pid\_t pid = 0; // process identifier

int high; // to store descending order from input

int low; // to store ascending order from input

int k = 1;

if (argc == 1) {

fprintf(stderr,"Usage: ./hw1 <starting value>\n");

return -1;

}

n = atoi(argv[1]); // n is the input starting value

//input must be positive number.

if(n < 0) {

printf("Number must be positive");

return -1;

}

// create pipe

if (pipe(fd) == -1) {

fprintf(stderr, "Pipe failed");

return -1;

}

//call fork; creates new process

pid = fork();

// wrong value

if(pid < 0) {

fprintf(stderr,"Failed");

return -1;

}

//child process; Kaprekar's Operation algorithm

else if (pid ==0){

//495 is Kaprekar's constant number when the number is 3 digit.

while (n !=495) {

int array[3] = {0,0,0}; //define array for 3 digit

int m = 0;

int i = 0;

int j = 0;

int temp;

temp = n; //store input value

// divide the input number by each digit.

// 1th digit, 10th digit 100th digit.

while (temp > 0){

array[m] = temp % 10;

temp = temp / 10;

m++;

}

// sort the digits by ascending order

for (i = 0; i < m; i++){

for (j = 0; j < m-1; j++){

if (array[j] > array[j+1]){

temp = array[j];

array[j] = array[j+1];

array[j+1] = temp;

}

}

}

high = array[0] + array[1] \* 10 + array[2] \* 100;

low = array[0] \* 100 + array[1] \* 10 + array[2];

// n is the result

n = high - low;

//pipe cannot transfer numbers, thus change the

//type from int to string

sprintf(buff, "%d", n);

printf("Child process %d: %dth: %d - %d = %d\n", pid, k, high, low, n);

k++; //count the number of while

}

//I will send result from child to parent

//Therefore close read pipe and open write pipe

close(fd[READ\_END]);

write(fd[WRITE\_END],buff,25);

}

// parent process

else {

wait (NULL); //waiting the child process

close(fd[WRITE\_END]); //close write pipe

read(fd[READ\_END],buff,BUFFER\_SIZE); //open read pipe

printf("Result from Child process %d: %s\n", pid, buff);

}

return 0;

}