BILKENT UNIVERSITY

CS 342 – OPERATING SYSTEMS

PROJECT #1 REPORT

Deniz Yüksel

21600880

Table of Comparisons

N	M	character-count	read-call count	user time(ms)	system time(ms)	real time (ms)
50	100000	100001	2001	0.00	0.00	0.01
50	1000000	1000001	20001	0.01	0.02	0.07
50	10000000	10000001	200001	-	-	-
100	50000	50001	501	0.00	0.00	0.00
100	100000	100001	1001	0.00	0.00	0.01
100	1000000	1000001	10001	0.02	0.01	0.05
1000	1000000	1000001	1001	0.01	0.00	0.02
1000	5000000	5000001	5001	0.07	0.02	0.10

Interpretations

My computer’s specifications are really basic and ubuntu 18.04 runs really slow in my machine. I think this may be the cause of for some of my experiments to make me wait for so long.

However, I discovered that generally in the table, the character-count and read-count are always linear. As the byte reading rate, that is named by N in this assignment increases, the real time elapsed decreases naturally because it reads more bytes per second. As the read data gets larger, user time tends to increase more than the system time. However their total are always the real time value.

Conclusions & Outputs

I wanted to include an output that is rather unusual but needs to be commented on. As honesty is the best policy, after some huge values I get an error saying that I receive and exit code of 11. I receive page faults. I think

maybe because I am not using dynamic allocation and my checking at a certain point for strings is faulty. But I am sure my pipes work correctly. Here is an output to be noticed :

>>>Enter command:time ./producer 50000000 | ./consumer 50000000

Command terminated by signal 11

0.00user 0.00system 0:00.22elapsed 0%CPU (0avgtext+0avgdata 1024maxresident)k

0inputs+0outputs (0major+51minor)pagefaults 0swaps

Code for bilshell.c, consumer.c, producer.c:

// bilshell.c

```
#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/wait.h>


#define clear() printf("\033[H\033[J")

#define MAXLETTERS 4096


int MAX_INPUT = 1024;

int MAX_PARAM = 128;


void printInitShell()

{

 clear();

 printf("Welcome to bilshell! \n");

}
```

```
void askPrompt()
{
printf(">>Enter command: ");

}

void executeOrderSingle(char** cmd)
{
pid_t pid = fork();
if( pid == -1){
printf("\nFailed to fork...");
return;
}
else if( pid == 0)
{
int i = 0;
execvp(cmd[0], cmd);
}
else
{
wait(NULL);
return;
}
}

void executeOrderPiped(char** cmd1, char** cmd2, int N)
{
int pipe1[2];
```

```
int pipe2[2];
pid_t pid1, pid2;

//create pipes.
pipe(pipe1);
pipe(pipe2);

pid1 = fork();
if( pid1 == -1){
printf("\nFailed to fork...");
return;
}
else if( pid1 == 0){//if we are in child1
close(pipe1[0]);
close(pipe2[0]);
close(pipe2[1]);
dup2(pipe1[1], 1);
execvp(cmd1[0], cmd1);
exit(0);
}
else{
//going to fork for the second child here!
pid2 = fork();
if( pid2 == 0) //child2
{
close(pipe1[0]);
close(pipe1[1]);
close(pipe2[1]);
```

```
dup2(pipe2[0],0);
execvp(cmd2[0], cmd2);
exit(0);
}
else //in parent
{
close(pipe2[0]);
close(pipe1[1]);
char buffer[N];
int readByte;
int counter = 0;
int totalReadByte = 0;
do{
readByte = read(pipe1[0], buffer, N);
totalReadByte = totalReadByte + readByte;
counter++;
write(pipe2[1], buffer, readByte);
}while(readByte);
close(pipe2[1]);
close(pipe2[0]);
close(pipe1[1]);
close(pipe1[0]);

wait(NULL);
wait(NULL);
counter--;
printf("character-count: %d\n", totalReadByte);
printf("read-call-count: %d\n", counter);
```

```
}
}
}

void interactiveMode(char *argv[])
{
printf("Interactive mode...\n");
// N is the speed of processing.
int N = atoi(argv[1]);
char inputString[MAXLETTERS];
char* cmd[20];
int i;
int strCount;
int pipeLoc;
do{
printf(">>>Enter command:");
i = 0;
strCount = 0;
pipeLoc = -1;
fgets(inputString, MAXLETTERS, stdin);
/*Get rid of null char...
char afterScanInput[strlen(inputString)];
for( i = 0; i < strlen(inputString) - 1; i++){
afterScanInput[i] = inputString[i];
}
*/
```

```
 //STRTOK

 const char s[3] = " \n";
char* token;

  token = strtok(inputString, s);

 /* walk through other tokens */
  while( token != NULL ) {
   //printf( "%d\n", token );
   cmd[strCount++] = token;
    token = strtok(NULL, s);
  }
  cmd[strCount] = NULL;
  if( !cmd[0]){
  printf("Please execute the program again with an input!!!\n");
  continue;
 }

/* STRTOK_R
char* token;
 char* rest = inputString;
while ((token = strtok_r(rest, " ", &rest))){
   cmd[strCount++] = token;
 }
 cmd[strCount] = NULL;
 */
 // I have read the whole command. Now I will find the pipe.
```

```
  for( i = 0; cmd[i]; i++){
  if( strcmp(cmd[i], "|") == 0){
pipeLoc = i;
  }
  }
//printf("%d ", pipeLoc);
if(strcmp(cmd[0], "exit") == 0)
exit(1);
if( pipeLoc > 0)
{
//there is a pipe then.
char* cmd1[20];
char* cmd2[20];
for( i = 0; i < pipeLoc; i++){
cmd1[i] = cmd[i];
//printf("%s\n", cmd1[i]);
}
for( i = pipeLoc + 1; i < strCount; i++){
cmd2[i - (pipeLoc + 1)] = cmd[i];
//printf("%s\n", cmd2[i-pipeLoc-1]);
}
/*
for(i = 0; cmd1[i]; i++) {
system(strcat("echo ", cmd1[i]));
}

for(i = 0; cmd2[i]; i++) {
```

```
system(strcat("echo ", cmd2[i]));
}
*/
executeOrderPiped(cmd1, cmd2, N);
}

else
{
//there is no pipe.
executeOrderSingle(cmd);
}
}while(1);
}

void batchMode(char *argv[])
{
printf("Batch mode. Processing your file... \n");
char* fileName = argv[2];
FILE* fp;
fp = fopen( fileName, "r");
int N = atoi(argv[1]);
char inputString[MAXLETTERS];
char* cmd[20];
while( fgets(inputString, N, fp)){
int i = 0;
int strCount = 0;
int pipeLoc = -1;
```

```
const char s[3] = " \n";
  char* token;

  token = strtok(inputString, s);

  /* walk through other tokens */
  while( token != NULL ) {
    //printf( "%d\n", token );
   cmd[strCount++] = token;
    token = strtok(NULL, s);
  }
cmd[strCount] = NULL;
if(strcmp(cmd[0], "exit") == 0)
exit(1);

  // I have read the whole command. Now I will find the pipe.
  for( i = 0; i < strCount; i++){
  if( strcmp(cmd[i], "|") == 0){
pipeLoc = i;
  }
  }
if( pipeLoc > 0)
{
//there is a pipe then.
char* cmd1[20];
char* cmd2[20];
for( i = 0; i < pipeLoc; i++){
```

```
cmd1[i] = cmd[i];
//printf("%s\n", cmd1[i]);
}
for( i = pipeLoc + 1; i < strCount; i++){
cmd2[i - (pipeLoc + 1)] = cmd[i];
//printf("%s\n", cmd2[i-pipeLoc-1]);
}
executeOrderPiped(cmd1, cmd2, N);
}

else
{
executeOrderSingle(cmd);
}
}
fclose(fp);

}

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

if( argc == 2){
interactiveMode(argv);
}
else if( argc == 3){
batchMode(argv);
}
```

```
else{
printf("You miss N value, or you have entered too many arguments for this assignment.");
}

return 0;
}
//////////////////////////////////////////////////producer.c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <time.h>

#define N 1000
// N CHARACTERS WILL BE READ AT A TIME.
int main( int argc, char *argv[])
{
if( argc == 2)
{
srand(time(NULL));
char allChars[] = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
int M = atoi(argv[1]);
char charGroup[M];
int i = 0;
int charCounter = 0;
//edge case:
```

```
if(N > M)
{
for( i = 0; i < N ; i++)
{
char randomChar = allChars[rand() % (sizeof(allChars) - 1)];
charGroup[i] = randomChar;
}
write(1, charGroup, M);
printf("\n");
}
else{
//write random chars N by N...
for( charCounter = 0; charCounter < M; charCounter = charCounter + N)
{
for(i = 0; i < N; i++)
{
charGroup[i] = allChars[rand() % (sizeof(allChars) - 1)];
}
write(1, charGroup, N);
}
/*
int excess = M % N;
char remChars[excess];
for(i = 0; i < excess; i++) {
remChars[i] = allChars[rand() % (sizeof(allChars) -1)];
}
//write the excess chars here.
write(1, charGroup, excess);
```

```
*/
printf("\n");
}
}
return 0;
}
//////////////////////////////////////// consumer.c

#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/wait.h>

#define N 1000

int main(int argc, char* argv[])
{
if( argc == 2)
{
int M = atoi(argv[1]);
int readBytes = 0;
  char charGroup[N];

  while (readBytes < M) {
     readBytes = readBytes + read(0, charGroup, N);
   }
}
```

```
return 0;
}
```