**OS Unit 3 & 4 – Assignment**

**Sriram R PES1UG20CS435 Section : H Roll # : 16**

Source codes & outputs :

1. Finding the sum of an array using pipes.

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#define ARRAY\_SIZE 8

#define BUFFER\_SIZE 8 // should be greater than expected number of digits in sum

#define READ\_END 0

#define WRITE\_END 1

int main()

{

int numArray[] = { 1, 2, 3, 4, 5, 6, 7, 8 };

int sumChild = 0;

int sum = 0;

char write\_msg[BUFFER\_SIZE] = "";

char read\_msg[BUFFER\_SIZE] = "";

int fd[2];

pid\_t pid;

if (pipe(fd) == -1)

{

printf("pipe failure\n");

return 1;

}

pid = fork();

if (pid > 0) // parent process - calculates sum of first half and stores in pipe

{

for (int i = 0; i < (ARRAY\_SIZE / 2); i++) sum += numArray[i];

snprintf(write\_msg, BUFFER\_SIZE, "%d", sum); // converts integer to string (itoa not supported)

printf(".... parent process is storing %s in pipe ....\n", write\_msg);

close(fd[READ\_END]);

write(fd[WRITE\_END], write\_msg, strlen(write\_msg) + 1);

close(fd[WRITE\_END]);

wait(NULL);

}

else if (pid == 0) // child process - calculates sum of second half, adds to the sum in the pipe and prints

{

close(fd[WRITE\_END]);

read(fd[READ\_END], read\_msg, BUFFER\_SIZE);

for (int i = (ARRAY\_SIZE / 2); i < ARRAY\_SIZE; i++) sumChild += numArray[i];

sumChild += atoi(read\_msg);

printf("child process summation : %d\n", sumChild);

close(fd[READ\_END]);

}

else

{

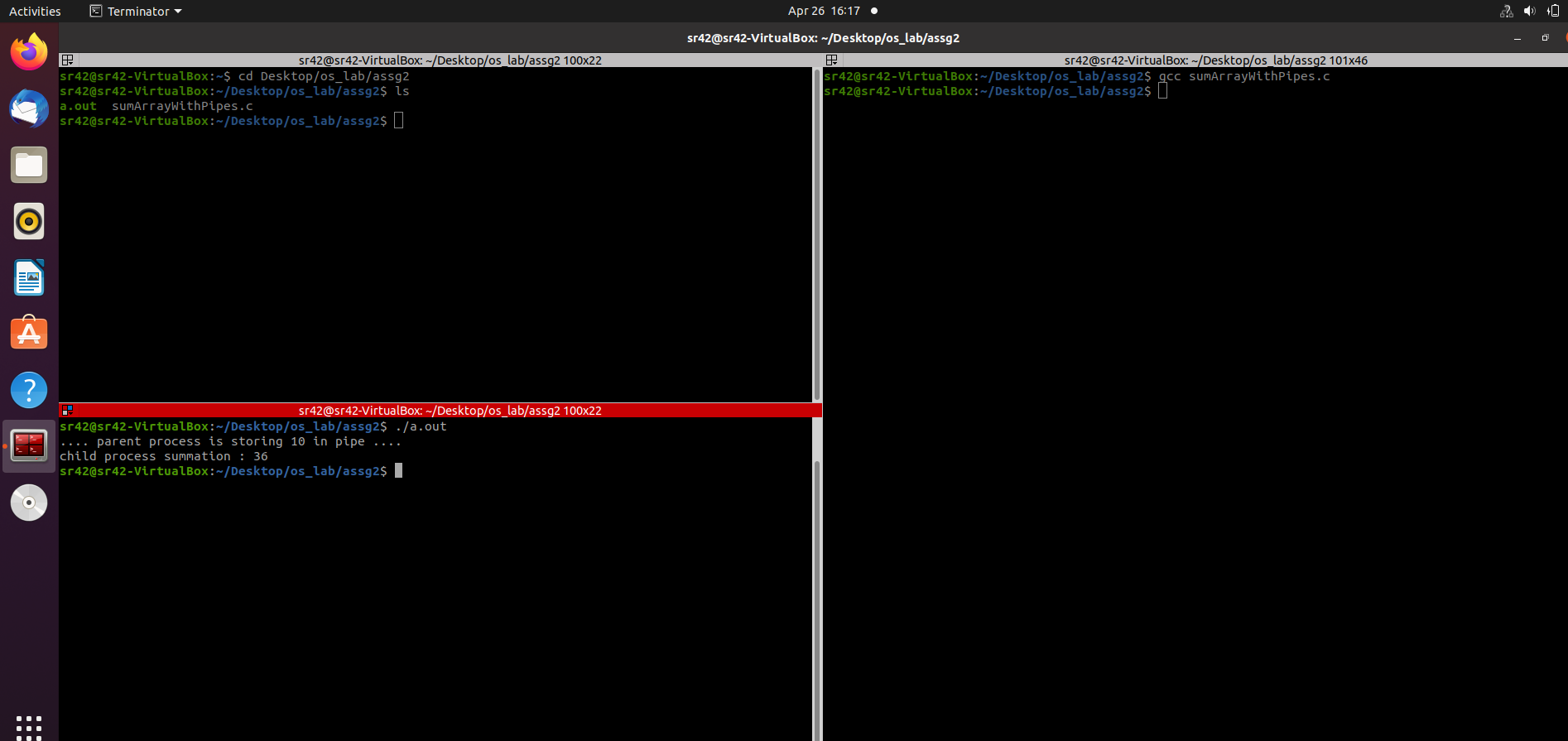
printf("fork failure\n");

return 1;

}

return 0;

}



1. Simulation of segmentation on C.

#include<stdio.h>

#include <stdlib.h>

struct list

{

int seg;

int base;

int limit;

struct list\* next;

}\*p;

void insert(struct list\* q, int base, int limit, int seg)

{

if (p == NULL)

{

p = malloc(sizeof(struct list));

p->limit = limit;

p->base = base;

p->seg = seg;

p->next = NULL;

}

else

{

while (q->next != NULL) q = q->next;

q->next = malloc(sizeof(struct list));

q->next->limit = limit;

q->next->base = base;

q->next->seg = seg;

q->next->next = NULL;

}

}

int find(struct list\* q, int seg)

{

while (q->seg != seg) q = q->next;

return q->limit;

}

int search(struct list\* q, int seg)

{

while (q->seg != seg) q = q->next;

return q->base;

}

int main()

{

p = NULL;

int seg, offset, limit, base, c, s, physical;

printf("Segmentation table input : \n");

printf("Input -1 as segment value for termination ...\n\n");

do

{

printf("Input segment number : ");

scanf("%d", &seg);

if (seg != -1)

{

printf("Input base value : ");

scanf("%d", &base);

printf("Input segment limit value : ");

scanf("%d", &limit);

insert(p, base, limit, seg);

}

} while (seg != -1);

printf("Input offset : ");

scanf("%d", &offset);

printf("Input segment number : ");

scanf("%d", &seg);

c = find(p, seg);

s = search(p, seg);

if (offset < c)

{

physical = s + offset;

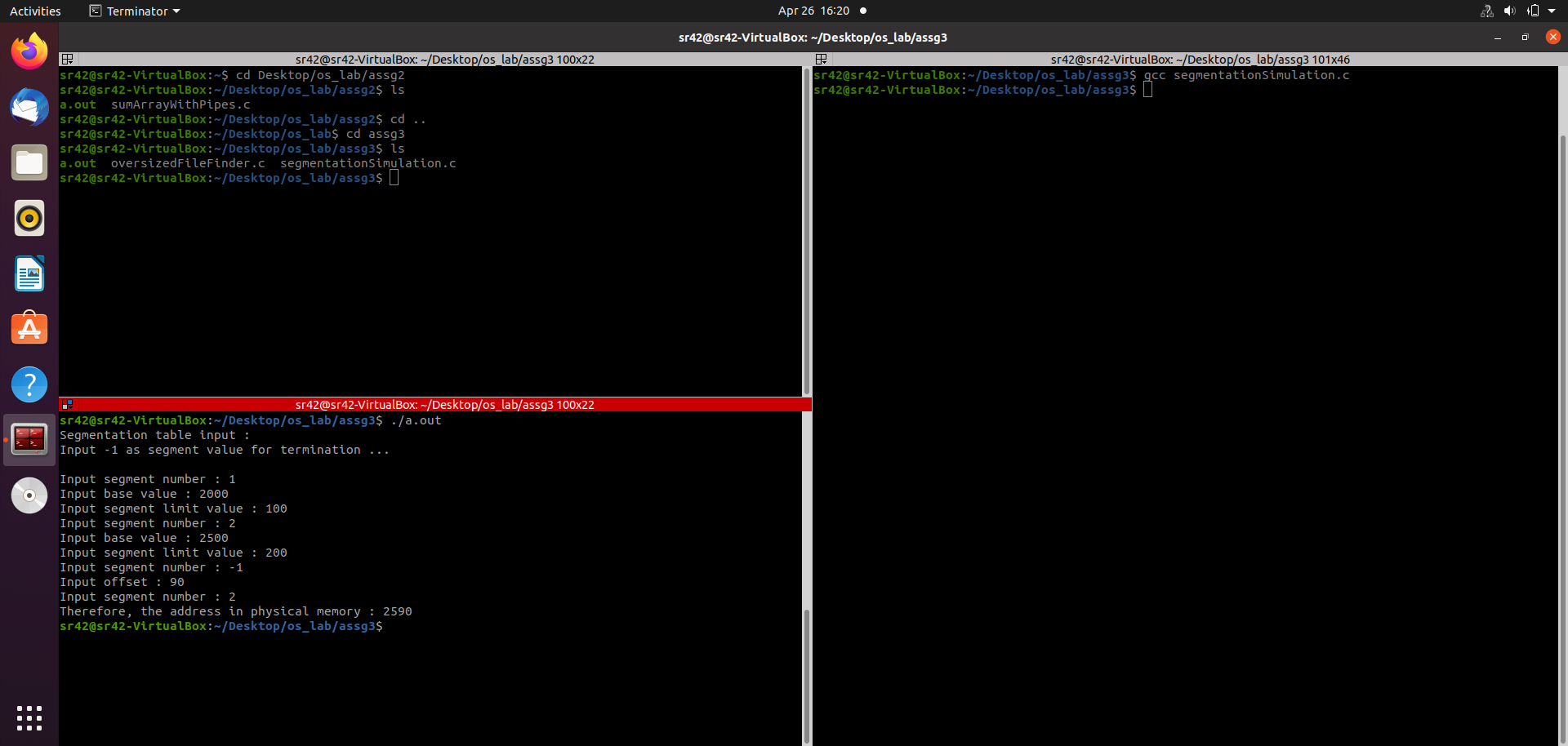
printf("Therefore, the address in physical memory : %d\n", physical);

}

else printf("error");

return 0;

}



1. Finding all the oversized files in a directory.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <dirent.h>

#include <sys/stat.h>

#include <time.h>

int main()

{

DIR\* d;

struct dirent\* dir;

struct stat stats;

long dt;

char dirname[256];

printf("Input directory: ");

fflush(stdout);

scanf("%s", dirname);

d = opendir(dirname);

int size;

printf("Input size: ");

scanf("%d", &size);

if (!d)

{

printf("error\n");

exit(0);

}

while ((dir = readdir(d)))

{

if (stat(dir->d\_name, &stats) == 0)

{

dt = stats.st\_size;

printf("%ld\n", dt);

if (dt > size) printf("File: %s\n", dir->d\_name);

}

else printf("Couldn't read file properties of file: %s\n ", dir->d\_name);

}

closedir(d);

return 0;

}

/\*

// alternate method – this one looks WAYY better but apparently using system() wasn’t allowed. I included it’s output in the second screenshot anyways

#include <stdio.h>

#include <stdlib.h>

int main()

{

char \*command = (char\*)malloc(256 \* sizeof(char));

char \*path = (char\*)malloc(256 \* sizeof(char));

int size;

printf("Input directory path : ");

scanf("%s", path);

printf("Input the desired file size : ");

scanf("%d", &size);

sprintf(command, "find %s -size +%dk", path, size);

system(command);

return 0;

}

\*/

