COMP304 Assignment 1 Report

Mert Erdem

83078

Spring 2024

Part 1/A:

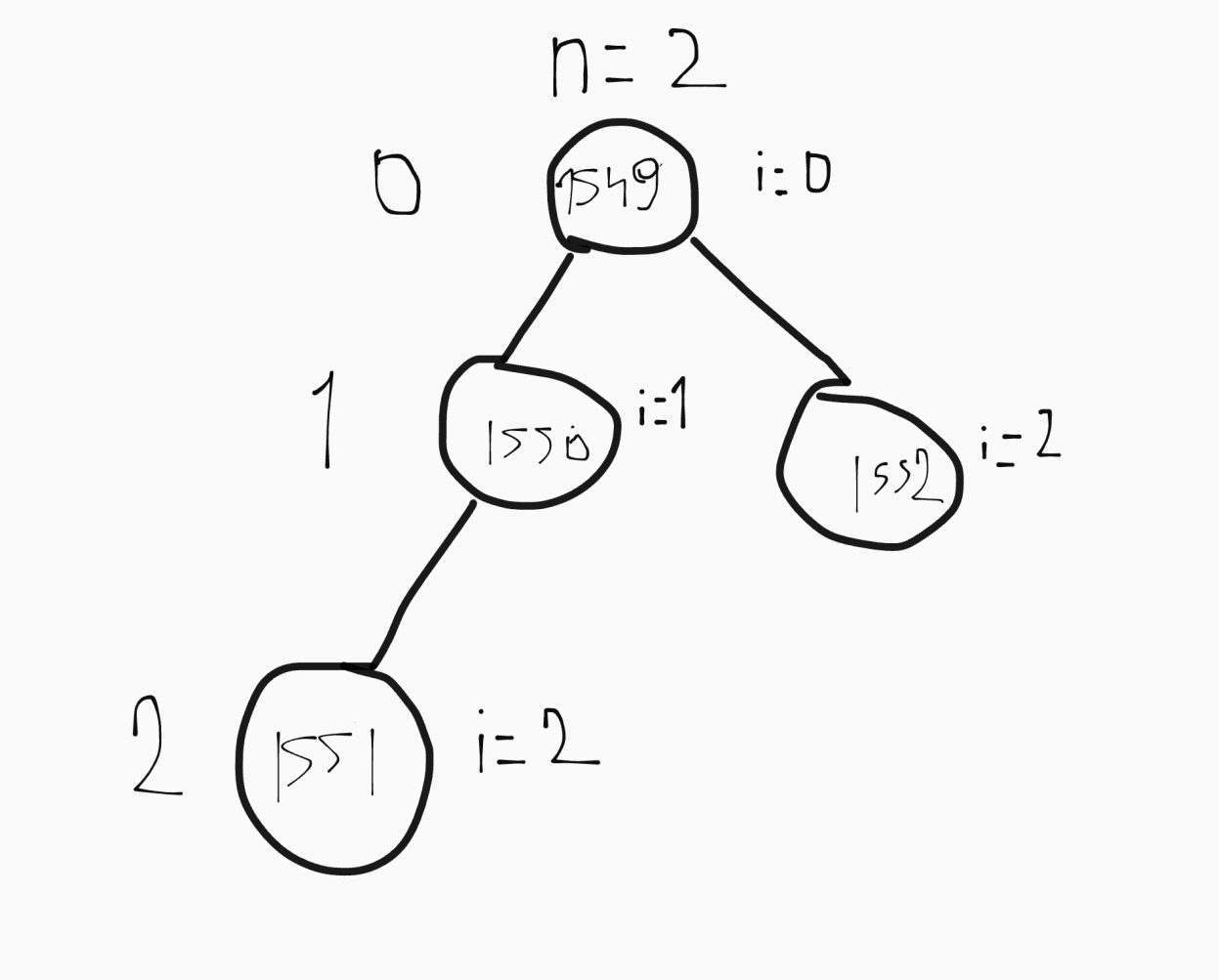
Code Explanation : This simple program takes the number of iterations from the command line and uses a for loop to create processes.

```c
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>

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


        if (argc != 2)//ensure command line arguments are entered properly.
        {
                printf("Please enter a single number as an argument to denote the number of iterations.");
                return 1;
        }


        int x = atoi(argv[1]);
        int i = 0;
        int level = 0;

        printf("Main Process ID: %d, level: %d\n", getpid(), level);

        for (int i = 0; i < x; i++)
        {

                int val = fork();
                if (val == 0)//this block will only be executed in the child process.
                {
                        level++;
                        printf("Process ID: %d, Parent ID: %d, Level: %d\n", getpid(), getppid(), level);
                }
                else
                {
                        wait(NULL);
                }

        }

        return 0;

}
~
~
~
~
```

Output:

```
merdem22@merdem22:~/COMP304/COMP304-assg1$ ./p1a 2
Main Process ID: 1549, level: 0
Process ID: 1550, Parent ID: 1549, Level: 1
Process ID: 1551, Parent ID: 1550, Level: 2
Process ID: 1552, Parent ID: 1549, Level: 1
merdem22@merdem22:~/COMP304/COMP304-assg1$ ./p1a 3
Main Process ID: 1553, level: 0
Process ID: 1554, Parent ID: 1553, Level: 1
Process ID: 1555, Parent ID: 1554, Level: 2
Process ID: 1556, Parent ID: 1555, Level: 3
Process ID: 1557, Parent ID: 1554, Level: 2
Process ID: 1558, Parent ID: 1553, Level: 1
Process ID: 1559, Parent ID: 1558, Level: 2
Process ID: 1560, Parent ID: 1553, Level: 1
merdem22@merdem22:~/COMP304/COMP304-assg1$ ./p1a 5
Main Process ID: 1561, level: 0
Process ID: 1562, Parent ID: 1561, Level: 1
Process ID: 1563, Parent ID: 1562, Level: 2
Process ID: 1564, Parent ID: 1563, Level: 3
Process ID: 1565, Parent ID: 1564, Level: 4
Process ID: 1566, Parent ID: 1565, Level: 5
Process ID: 1567, Parent ID: 1564, Level: 4
Process ID: 1568, Parent ID: 1563, Level: 3
Process ID: 1569, Parent ID: 1568, Level: 4
Process ID: 1570, Parent ID: 1563, Level: 3
Process ID: 1571, Parent ID: 1562, Level: 2
Process ID: 1572, Parent ID: 1571, Level: 3
Process ID: 1573, Parent ID: 1572, Level: 4
Process ID: 1574, Parent ID: 1571, Level: 3
Process ID: 1575, Parent ID: 1562, Level: 2
Process ID: 1576, Parent ID: 1575, Level: 3
Process ID: 1577, Parent ID: 1562, Level: 2
Process ID: 1578, Parent ID: 1561, Level: 1
Process ID: 1579, Parent ID: 1578, Level: 2
Process ID: 1580, Parent ID: 1579, Level: 3
Process ID: 1581, Parent ID: 1580, Level: 4
Process ID: 1582, Parent ID: 1579, Level: 3
Process ID: 1583, Parent ID: 1578, Level: 2
Process ID: 1584, Parent ID: 1583, Level: 3
Process ID: 1585, Parent ID: 1578, Level: 2
Process ID: 1586, Parent ID: 1561, Level: 1
Process ID: 1587, Parent ID: 1586, Level: 2
Process ID: 1588, Parent ID: 1587, Level: 3
Process ID: 1589, Parent ID: 1586, Level: 2
Process ID: 1590, Parent ID: 1561, Level: 1
Process ID: 1591, Parent ID: 1590, Level: 2
Process ID: 1592, Parent ID: 1561, Level: 1
merdem22@merdem22:~/COMP304/COMP304-assg1$ _
```

Simple Visualization as a process tree (for n=2):



Part 1/B:

Code explanation: This simple program creates a child process and this child process will exit immediately, causing it to turn into a zombie.

```c
#include <stdio.h>
#include <stdlib.h>


int main(){


        int val = fork();
        if (val == 0)
        {
                //the child will terminate immediately.
                exit(EXIT_SUCCESS);
        }
        else
        {
                sleep(5);//the parent will remain running for 5 seconds, but its child has terminated itself turning into a zombie.
        }
        return 0;
}
~
~
~
~
~
```

Output: It can be seen that the p1b<defunct> is in state Z, indicating that it is a zombie.

```
merdem22@merdem22:~/COMP304/COMP304-assg1$ ./p1b
^Z
[1]+  Stopped                 ./p1b
merdem22@merdem22:~/COMP304/COMP304-assg1$ ps -l
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S  1000    1017     701  0  80   0 -  2091 do_wai tty1     00:00:00 bash
0 T  1000    1609    1017  0  80   0 -   513 do_sig tty1     00:00:00 p1b
1 Z  1000    1610    1609  0  80   0 -     0 -      tty1     00:00:00 p1b <defunct>
0 R  1000    1611    1017  0  80   0 -  2354 -      tty1     00:00:00 ps
merdem22@merdem22:~/COMP304/COMP304-assg1$
```

Part 2:

The code: The program works by forking n processes, and using execvp() to load the program entered in the command line, the program also uses gettimeofday() to measure the running times of the processes.

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>

int main(int argc, char *argv[])
{
        if (argc != 3)
        {
                printf("Please enter n and the command_name properly.");//ensuring proper command line arguments..
                return 1;
        }
        //get input.
        int n = atoi(argv[1]);
        char *command_name = argv[2];

        int val;
        double durations[n];

        struct timeval start_time, end_time;
        double duration;

        for (int i = 0; i < n; i++)
        {
                gettimeofday(&start_time, NULL); //start the chronometer.
                val = fork();
                if (val == 0)
                {       //forwarding to dev/null
                        int null_fd = open("/dev/null", O_WRONLY);
                        if(dup2(null_fd, STDOUT_FILENO) == -1)
                        {
                                perror("dup");
                                exit(NULL);
                        }
                        close(null_fd);



                        execvp(command_name, (char *)NULL); //runs the command specified with no arguments.
                        exit(NULL);//since the child exits after execution, they won't be able to create processes themselves.

                }
                else
                {
                        wait();//the parent will wait for its children to finish.
                }
                gettimeofday(&end_time, NULL);//stop the chronometer.
"p2.c" 64L, 1381B
```

```c
                }
                gettimeofday(&end_time, NULL);//stop the chronometer.
                duration = (double)(end_time.tv_sec - start_time.tv_sec) * 1000.0 + (double)(end_time.tv_usec - start_time.tv_usec) / 1000.0;
                durations[i] = duration;


        }

        for (int i = 0; i < n; i++)
        {
                printf("It took process %d, %lf miliseconds.\n", i, durations[i]);
        }



        return 0;
}
```

The output:

```
merdem22@merdem22:~/COMP304/COMP304-assg1$ ./p2 5 ls
It took process 0, 1.409000 miliseconds.
It took process 1, 1.024000 miliseconds.
It took process 2, 0.841000 miliseconds.
It took process 3, 0.926000 miliseconds.
It took process 4, 1.964000 miliseconds.
merdem22@merdem22:~/COMP304/COMP304-assg1$ ./p2 3 pwd
It took process 0, 4.701000 miliseconds.
It took process 1, 0.759000 miliseconds.
It took process 2, 1.196000 miliseconds.
merdem22@merdem22:~/COMP304/COMP304-assg1$ _
```

Part 3:

The code: the program works by forking n processes, then determining the indices that will be searched in the array entered from stdin for each process, the program also stores the pids of each process to kill them if a match is found. Killing is done with kill() and SIGKILL signal.

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


#define MAX_NUMBERS 1000


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

        //set up the arguments
        int x = atoi(argv[1]);
        int processCount = atoi(argv[2]);
        pid_t childrenPIDS[processCount]; //get the pids from this array once its time to kill the children.

        int numbers[MAX_NUMBERS];
        int numCount = 0;

        if (argc != 3)
        {
                //ensure propper command line arguments.
                printf("Please enter x and n as command line arguments.");
                return 1;
        }

        //assign numbers to array from stdin.
        while (numCount < MAX_NUMBERS && scanf("%d", &numbers[numCount]) == 1)
        {
                numCount++;
                while(getchar() != '\n');

        }

        int portionSize = numCount/processCount;
        int remainder = numCount % processCount;

        for (int i = 0; i < processCount; i++)
        {
                int val = fork();
                childrenPIDS[i] = val;
                if (val == 0)
                {

                        int startIndex = portionSize * i;
                        int endIndex;
                        if (i == processCount - 1) //setting up the borders of the array portion.
                        {
```

```
                        if (i == processCount - 1) //setting up the borders of the array portion.
                        {
                                endIndex = numCount;
                        }
                        else
                        {       endIndex = startIndex + portionSize + (i < remainder ? 1 : 0);
                        }
                        for (int j = startIndex; j < endIndex; j++) //search the portion.
                        {
                                if (numbers[j] == x)
                                {
                                        printf("Process with ID: %d has found X: %d at index: %d\n", getpid(), x, j);
                                        exit(0);
                                }

                        }
                        //not found
                        printf("Process with ID: %d failed to find a match.\n", getpid());
                        exit(1);
                }
                else
                {       int status;
                        waitpid(childrenPIDS[i], &status, 0); //wait for the current child to finish.
                        if (WIFEXITED(status) && WEXITSTATUS(status) == 0) //if the process exits with code 0.
                        {
                                for (int j = 0; j < i+1; j++)//kill the remaining processes.
                                {
                                        if (j!=i)
                                        {
                                                kill(childrenPIDS[j], SIGKILL);
                                                printf("Process with ID: %d is killed.\n", childrenPIDS[j]);
                                        }
                                }
                                break;
                        }

                }
        }

        return 0;
}
```

The output:

```
merdem22@merdem22:~/COMP304/COMP304-assg1$ shuf -i 1-1000 | ./p3 1000 5
Process with ID: 46921 failed to find a match.
Process with ID: 46922 failed to find a match.
Process with ID: 46923 failed to find a match.
Process with ID: 46924 failed to find a match.
Process with ID: 46925 has found X: 1000 at index: 905
Process with ID: 46921 is killed.
Process with ID: 46922 is killed.
Process with ID: 46923 is killed.
Process with ID: 46924 is killed.
merdem22@merdem22:~/COMP304/COMP304-assg1$ _
```

```
merdem22@merdem22:~/COMP304/COMP304-assg1$ shuf -i 1-1000 | ./p3 2 5
Process with ID: 46963 failed to find a match.
Process with ID: 46964 has found X: 2 at index: 381
Process with ID: 46963 is killed.
merdem22@merdem22:~/COMP304/COMP304-assg1$ _
```

```
merdem22@merdem22:~/COMP304/COMP304-assg1$ shuf -i 1-1000 | ./p3 98 3
Process with ID: 46889 failed to find a match.
Process with ID: 46890 failed to find a match.
Process with ID: 46891 has found X: 98 at index: 720
Process with ID: 46889 is killed.
Process with ID: 46890 is killed.
merdem22@merdem22:~/COMP304/COMP304-assg1$
```