TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

Pulchowk Campus

Lab No: **2**

A lab report on:

Process Concepts

Submitted By:
Bibek Basyal
075BCT097

Submitted To:
Department of Electronics
and Computer Engineering

February 6, 2022

Process Concepts

Theory:

A process is the unit of work in modern time-sharing systems. A system has a collection of processes (User processes as well as system processes). All these processes can execute concurrently with the CPU multiplexed among them.

Fork:

The fork system call creates a new process. When a program calls fork(), there will be two copies of the programs running simultaneously. Each copy can do what it desires independent of the other.

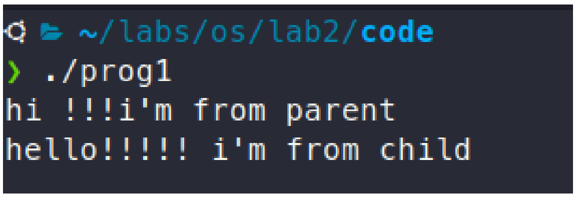
The fork() function returns the childs process id to the parent. It returns 0 to the child and returns a negative number in case of failure.

Program 1

```
#include <stdio.h>

main() {
    if(!fork()) {
        printf("hello!!!!! i’m from child\n");
    } else {
        printf("hi !!!i’m from parent\n");
    }
}
```

- **Output**



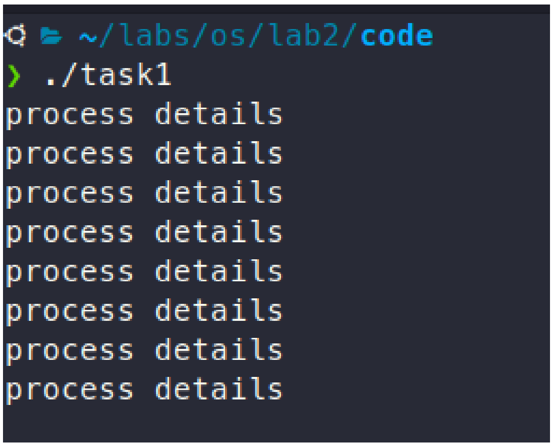
```
~/labs/os/lab2/code
❯ ./prog1
hi !!!i'm from parent
hello!!!!! i'm from child
```

Task 1

```
#include <stdio.h>

main() {
    fork();
    fork();
    fork();
    printf("process details\n");
}
```

- **Output**



```
~/labs/os/lab2/code
❯ ./task1
process details
process details
process details
process details
process details
process details
process details
process details
```

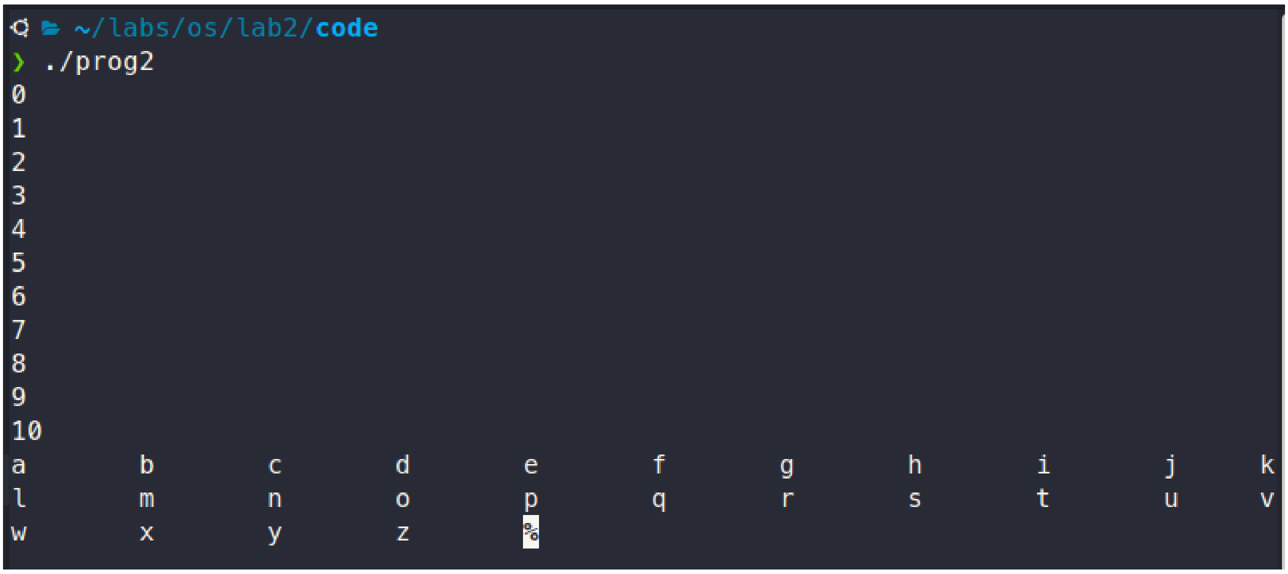
Program 2

```
#include <stdio.h>

#define DEL1 50
#define DEL2 100

main() {
    int i, d;
    char c;
    if (!fork()) {
        for (c = ’a’; c <= ’z’; c++) {
            printf("%c\t", c);
            fflush(stdout);
            for (d = 0; d < DEL1; d++);
        }
        exit(0);
    } else {
        for (i = 0; i <= 10; i++) {
            printf("%i\n", i);
            fflush(stdout);
            for (d = 0; d <= DEL2; d++);
        }
        exit(0);
    }
}
```

- **Output**



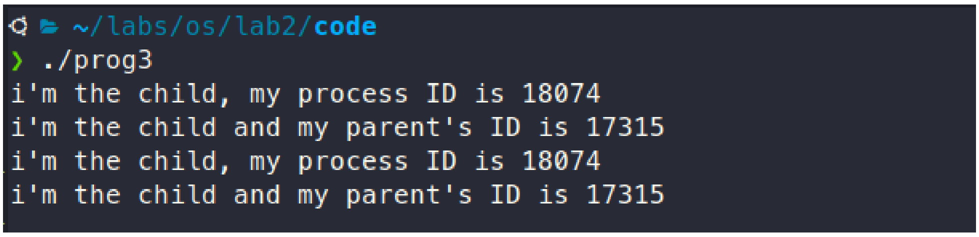
```
~/labs/os/lab2/code
❯ ./prog2
0
1
2
3
4
5
6
7
8
9
10
a       b       c       d       e       f       g       h       i       j       k
l       m       n       o       p       q       r       s       t       u       v
w       x       y       z       %
```

Program 3

```
#include <stdio.h>

main() {
    int pid;
    int fork();
    if(pid==0) {
        printf("i’m the child, my process ID is %d\n", getpid());
        printf("i’m the child and my parent’s ID is %d\n", getppid());
        sleep(20);
        printf("i’m the child, my process ID is %d\n", getpid());
        printf("i’m the child and my parent’s ID is %d\n", getppid());
    } else {
        printf("i’m the parent, my process ID is %d\n", getpid());
        printf("the parent’s process ID is %d\n", getppid());
    }
}
```

- **Output**



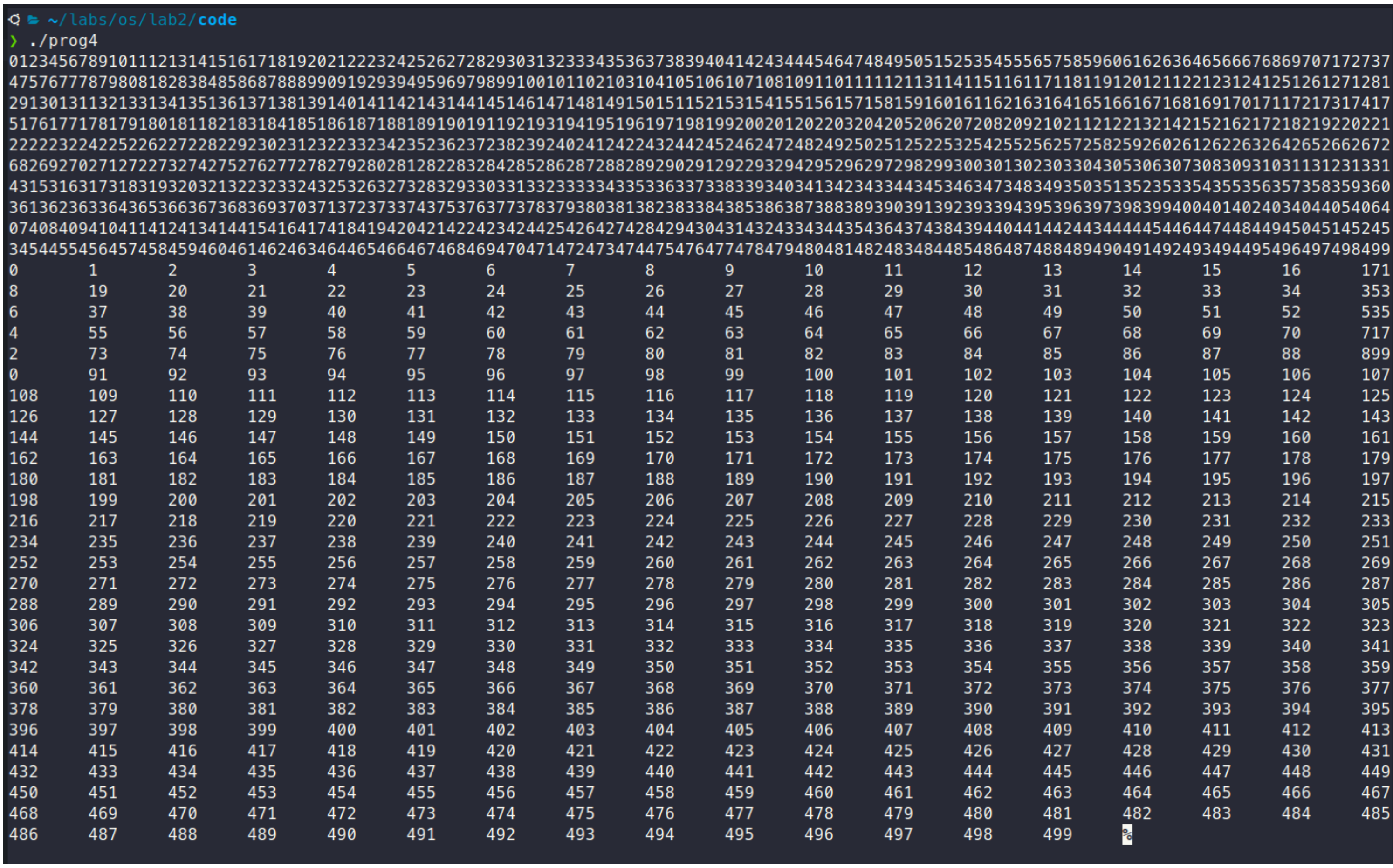
```
~/labs/os/lab2/code
❯ ./prog3
i'm the child, my process ID is 18074
i'm the child and my parent's ID is 17315
i'm the child, my process ID is 18074
i'm the child and my parent's ID is 17315
```

Program 4

```
#include <stdlib.h>

main() {
    int i = 0, j = 0, pid;
    pid = fork();
    if(pid==0) {
        for(i=0;i<500;i++)
        printf("%d\t", i);
    } else {
        if(pid>0) {
            for(j=0;j<500;j++)
                printf("%d",j);
        }
    }
}
```

- **Output**



```
~/labs/os/lab2/code
❯ ./prog4
0123456789101112131415161718192021222324252627282930313233343536373839404142434445464748495051525354555657585960616263646566676869707172737
4757677787980818283848586878889909192939495969798991001011021031041051061071081091101111121131141151161171181191201211221231241251261271281
2913013113213313413513613713813914014114214314414514614714814915015115215315415515615715815916016116216316416516616716816917017117217317417
5176177178179180181182183184185186187188189190191192193194195196197198199200201202203204205206207208209210211212213214215216217218219220221
2222232242252262272282292302312322332342352362372382392402412422432442452462472482492502512522532542552562572582592602612622632642652662672
6826927027127227327427527627727827928028128228328428528628728828929029129229329429529629729829930030130230330430530630730830931031131231331
4315316317318319320321322323324325326327328329330331332333334335336337338339340341342343344345346347348349350351352353354355356357358359360
3613623633643653663673683693703713723733743753763773783793803813823833843853863873883893903913923933943953963973983994004014024034044054064
0740840941041141241341441541641741841942042142242342442542642742842943043143243343443543643743843944044144244344444544644744844945045145245
3454455456457458459460461462463464465466467468469470471472473474475476477478479480481482483484485486487488489490491492493494495496497498499
0       1       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      171
8       19      20      21      22      23      24      25      26      27      28      29      30      31      32      33      34      353
6       37      38      39      40      41      42      43      44      45      46      47      48      49      50      51      52      535
4       55      56      57      58      59      60      61      62      63      64      65      66      67      68      69      70      717
2       73      74      75      76      77      78      79      80      81      82      83      84      85      86      87      88      899
0       91      92      93      94      95      96      97      98      99      100     101     102     103     104     105     106     107
108     109     110     111     112     113     114     115     116     117     118     119     120     121     122     123     124     125
126     127     128     129     130     131     132     133     134     135     136     137     138     139     140     141     142     143
144     145     146     147     148     149     150     151     152     153     154     155     156     157     158     159     160     161
162     163     164     165     166     167     168     169     170     171     172     173     174     175     176     177     178     179
180     181     182     183     184     185     186     187     188     189     190     191     192     193     194     195     196     197
198     199     200     201     202     203     204     205     206     207     208     209     210     211     212     213     214     215
216     217     218     219     220     221     222     223     224     225     226     227     228     229     230     231     232     233
234     235     236     237     238     239     240     241     242     243     244     245     246     247     248     249     250     251
252     253     254     255     256     257     258     259     260     261     262     263     264     265     266     267     268     269
270     271     272     273     274     275     276     277     278     279     280     281     282     283     284     285     286     287
288     289     290     291     292     293     294     295     296     297     298     299     300     301     302     303     304     305
306     307     308     309     310     311     312     313     314     315     316     317     318     319     320     321     322     323
324     325     326     327     328     329     330     331     332     333     334     335     336     337     338     339     340     341
342     343     344     345     346     347     348     349     350     351     352     353     354     355     356     357     358     359
360     361     362     363     364     365     366     367     368     369     370     371     372     373     374     375     376     377
378     379     380     381     382     383     384     385     386     387     388     389     390     391     392     393     394     395
396     397     398     399     400     401     402     403     404     405     406     407     408     409     410     411     412     413
414     415     416     417     418     419     420     421     422     423     424     425     426     427     428     429     430     431
432     433     434     435     436     437     438     439     440     441     442     443     444     445     446     447     448     449
450     451     452     453     454     455     456     457     458     459     460     461     462     463     464     465     466     467
468     469     470     471     472     473     474     475     476     477     478     479     480     481     482     483     484     485
486     487     488     489     490     491     492     493     494     495     496     497     498     499     %
```

Observations:

Program 1:

In this program, when a fork() is called, then it creates a duplicate process and returns 0 to it, which is a child process. In the parent process the OS returns the newly created process id. As the value of fork is different for different processes (i.e child arent parent) the child executes 'if' block and parent executes 'else' block. In this program, the order of execution of these two blocks are not deterministic as sometimes parent might can run first while at other times child can run first.

Task1:

Using getpid() and get ppid() syscalls to print the process id and parent process id in the second program. we see that each of the program prints aline. There are 8 processes spawned in total because there are 3 fork calls. Some processes have other processes as parent process id. Initially this is the case for all spawned processes but some parent process exit while their child has not exited. In such case, the parent becomes the 'init' process and the process id of init process is 1. So we see some process have parent id 1.

Program 2:

The given program is same as the program 1 but has complex logic. For both parent and child after the fork(). This program also handles the error case when the fork() returns a negative value. In this program the child process prints the letter 'a' through 'z' while the parent process prints number '0' to '9'. After each character is written to the print buffer, it is flushed to parent bufferring.

After printing each character, the child does nothing for DEL1 times and ~~child~~ parent does nothing for DEL2 times and the ~~parent~~ DEL1 and DEL2 can set as required. Depending upon ~~the outputs~~ these values the outputs can be in different orders.

Program 3:

The program forks() and parent and child both prints their own process Id and their parent's process id. Here if the parent has not exited while parent process id is as expected. When the parent sleeps and hence is alive. the child's parent process id matches that of the parent. But, when the paren't doesn't sleep and exits, the child is adopted by the init process and the child parent id becomes 1, which ~~be~~ is seen in the output.

Program 4:

The program is samer as before. When execeuted the parent and child both print numbers from 0 to 499 and exit. As there is no synchronization going on. The numbers from the processes are mixed. Thus, there are various possible sequences. The exact sequence depends upon the host machine where the program is running. Other synchronization systems are provided to predict the behaviour of the sequence to be printed.

Zombie process:

A zombie process is a process in its terminated state. When a child function has finished execution, it sends an exit status to its parent function, Until the parent function recieves and acknowledge the message, the child function remains in 'zombie state' meaning it has executed but not exited.

Orphan process:

An orphan process is a process whose parent process has exited through its child is still running. These processes are adopted by the init process on unix and the init process periodically calls ~~wat~~ wait() on its children.

Diccussion:

Thus, the fork() system call can be used to spawn a new duplicate process. The processes run independently of one another. The OS provides ways to synchronize the process via sysCalls and their semantics. The wait() sys. call is used when a parent progress has to wait for it's child to complete execution.

Conclusion:

Thus, in this lab we learnt about different concepts about the unix processes. we learnt about the fork(). and wait() system calls and the process life cycle.