# Lab #04

## CASE STATEMENT

For each case value, you can match a single string or multiple strings with | (hi|hello returns true if $1 equals hi or hello in third example), or you can use the * or ? patterns (what*). To make a default case that catches all possible values other than the case val-ues specified, use a single * as shown by the final case in the examples. The syntax of case statement is:

```
case          $variable     name in
pattern1) command.......command;;
pattern2) command    .......command;;
patternN) command.......command;;
 ) command......command;
;esac
```

Examples

```
#!/bin/bash
echo "Is it morning? Please   answer yes or no"
read timeofday

case "$timeofday" in
yes) echo "Good Morning";;
no ) echo "Good Afternoon"    ;;
y ) echo "Good Morning"    ;;
n ) echo "Good Afternoon"    ;;
)   echo "Sorry,  answer not  recognized"   ;;
esac
```

```
#!/bin/bash
case $1 in
bye )
    echo Fine,  bye.
    ;;
hi|hello)
    echo Nice to  see you.
    ;;
what  )
    echo Whatever.
    ;;
)
echo 'Huh?'
;;
esac
```

## 13 ARITHMETIC OPERATIONS

Arithmetic expansion and evaluation is done by placing an integer expression using the following format:

$$((expression))$$

$$(( n1+n2 ))$$

The shell script below includes some arithmetic operations.

```
# !/bin/ba
shclear
echo "hello world"
var1=10;
var2=20;
var3=$(( $var1 + $var2  ));
var4=$(( $var1    $var2  ));
var5=$(( $var1    $var2  ));
var6=$(( $var1 / $var2  ));
echo $var3;
echo $var4;
echo $var5;
echo $var6;
exit
```

## 14 LOOPS

There are two kinds of loops in the shell: for and while loops.

1. **For Loop**

   The basic syntax of for loop is:

   ```
   for variable_name in    {list of values  }
   do
        execute one for each item  in the  list until list is not finished
        repeat all statement  between do and done
   done
   ```

   Examples:

   ```
   #!/bin/bash
   for i in 1 2 3 4 5
   do
        echo "Welcome $i times"
   done
   ```

   ```
   #!/bin/ba
   shi=1
   for day in Mon Tues Wed Thu Fri
   do
        echo "Weekday $((i++)): $day"
   done
   ```

   Sample Output:
   Weekday 1 : Mon
   Weekday 2 : Tue
   Weekday 3 : Wed
   Weekday 4 : Thu

Weekday 5 : Fri

```bash
#!/bin/bash
#define interval in for loop
#the loop execute from 1 to 5
for i in {1..5}
do
    echo "Welcome $i times"
done
```

```bash
#!/bin/bash
#the loop execute from 0 to 10
#with a jump of 2 in each interation
for i in {0..10..2}
do
    echo "Welcome $i times"
done
```

2. **While Loop**
   The basic syntax of while loop is:

```bash
#!/bin/bash
while [ condition ]
do
    command_1
    command_2
    ..............
    command_N
done
```

Examples:

```bash
#!/bin/bash
echo "Enter password"
read trythis

while [ "$trythis" != "secret" ]; do
    echo "Sorry, try again"
    read trythis
done
```

```bash
#!/bin/bash
#set n to 1
n=1

#continue until n equals 5
while [ $n le 5 ]
do
    echo "Welcome $n times."
    n=$(( n+1 ))
done
```

Sample Output:
Welcome 1 times.
Welcome 2 times.
Welcome 3 times.
Welcome 4 times.
Welcome 5 times.

```bash
#!/bin/bash
counter=$1
factorial=1
while [ $counter gt 0 ] do

    factorial=$(($factorial  $counter ))
    counter=$(($counter 1   ))
done
echo $factorial
```

# Lab Manual 04
# Process Creation
## fork, wait, exit, getpid and getppid Unix System Calls

---

## 15 COMPILE AND EXECUTE A C PROGRAM

Write down the C code in a text editor of your choice and save the file using **.c** extension i.e. filename.c. Now through your terminal move to the directory where filename.c resides.

```
gcc firstprogram.c
```

It will compile the code and by default an executable named a.out is created.

```
gcc o firstprogram   firstprogram.c
```

If your file is named firstprogram.c then type -o firstprogram as the parameter to gcc. It would create an executable by the name firstprogram for the source code named firstpro-gram.c .
To execute the program simply write the following:

```
./a.out   OR    ./firstprogram
```

In case you have written the code in C++ saved using .cpp extension, compile the code using g++ as:

```
g++ filename.cpp  OR    g++   o exec_name filename.cpp
```

And execute as ./a.out or ./exec_name

## 16 PROCESS CREATION

When is a new process created?

1. System startup

2. Submit a new batch job/Start program

3. Execution of a system call by process

4. A user request to create a process

On the creation of a process following actions are performed:

1. Assign a unique process identifier. Every process has a unique process ID, a non-negative integer. Because the process ID is the only well-known identifier of a process that is always unique, it is used to guarantee uniqueness.

2. Allocate space for the process.

3. Initialize process control block.

4. Set up appropriate linkage to the scheduling queue.

## 17 FORK SYSTEM CALL

An existing process can create a new process by calling the fork function.

```
#include<unistd.
h>pid_tfork(void);
// Returns:    0 in  child,process ID ofchild in parent,    1 on error
```

The definition of the pid_t is given in <sys/types> include file and <unistd.h> contain the declaration of fork system call.

### IMPORTANT POINTS

1. The new process created by fork is called the child process.

2. This function is called once but returns twice. The only difference in the returns is that the return value in the child is 0, whereas the return value in the parent is the process ID of the new child.

3. Both the child and the parent continue executing with the instruction that follows the call to fork.

4. The child is a copy of the parent. For example, the child gets a copy of the parent's data space, heap, and stack. Note that this is a copy for the child; the parent and the child do not share these portions of memory.

5. In general, we never know whether the child starts executing before the parent, or vice versa. The order depends on the scheduling algorithm used by the kernel.
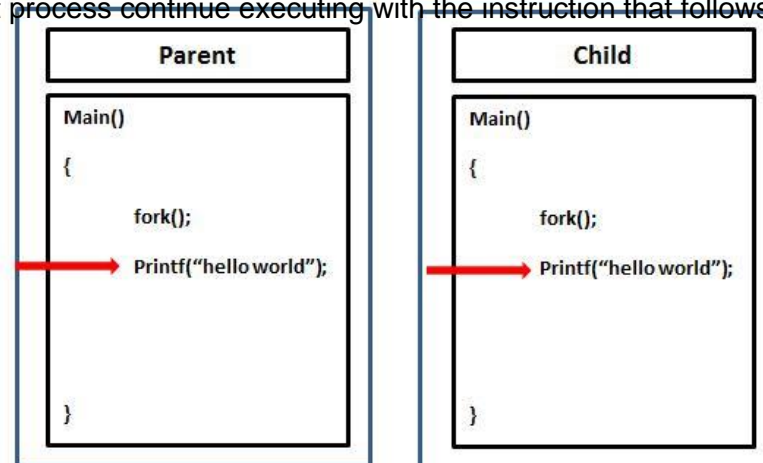
21 [The process will execute sequentially until it arrives at fork system call. This process will be named as parent process and the one created using fork() is termed as

**Parent**

```
Main()
{

    fork();

    Printf("hello world");



}
```
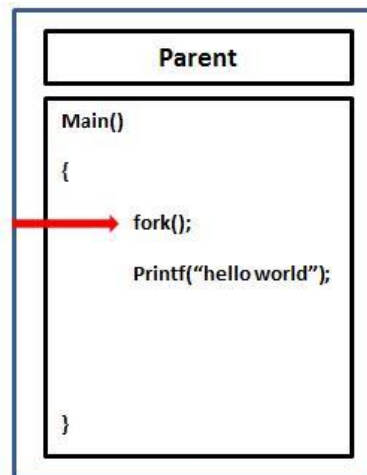
child process.]                                          [After fork system call both the child

and the parent process continue executing with the instruction that follows the call to

**Parent**

```
Main()
{

    fork();

    Printf("hello world");



}
```

**Child**

```
Main()
{

    fork();

    Printf("hello world");



}
```

fork.]

Figure 3: Parent and Child Process

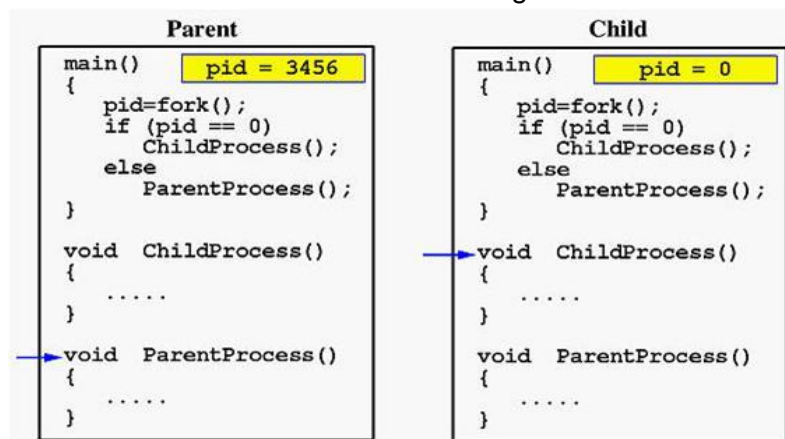21 [Fork system call returns child process ID (created using fork) in parent process and
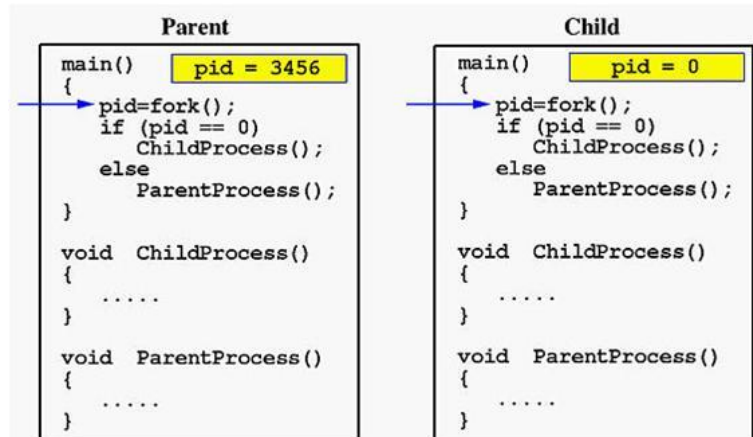


zero in child process]                                                                                           [We can execute different portions of code in parent and child process based on the return value of fork that is either zero or greater than



zero.]

Figure 4: Return Value of fork system call

## 18 EXAMPLES OF FORK()

1. Fork()'s Return Value

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

void main ( void ) {
        pid_t pid=fork();
        if(pid  == 0){ //This check will pass only for child process
                printf("I am Child  process \n");
        }
```

```c
        else if (pid > 0){ //   THIS Check will pass only for parent
                printf("I am Parent process \n");
        }

        else if (pid < 0){ //   if fork() fails
                printf("Error in Fork");
        }
}
```

2. Manipulating Local and Global Variables

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

int global=0;

int main()
{
        int status;
        pid_t child_pid;
        int local = 0;
        /   now create new process   /
        child_pid = fork();

        if (child_pid >= 0){  / fork   succeeded /
                if (child_pid == 0){
                        / fork() returns 0 for the child process /
                        printf("child process!\n");
                        // Increment    the local and global variables
                        local++;
                        global++;
                        printf("child PID  = %d, parent pid = %d\n",
                                    getpid(), getppid());
                        printf("\n child's local = %d, child's
                                    global = %d\n", local, global);
                }

                else{ / parent process /
                        printf("parent process!\n");
                        printf("parent PID = %d, child pid = %d\n",
                                    getpid(), child_pid);
                        int w=wait(&status);
                        //The      change in local and global variable
                        //in   child process should not reflect
                        //here in    parent process.
                        printf("\n Parent'z   local = %d, parent's global
                                    = %d\n", local, global);
                        printf("Parent says bye!\n");

                        exit(0); / parent exits /
                }
```

```
        }

            else{ / failure /
                   perror("fork");
                   exit(0);
            }
}
```

## 19 WAIT SYSTEM CALL

This function blocks the calling process until one of its child processes exits. wait() takes the address of an integer variable and returns the process ID of the completed process.

```
pid_t wait (int    status)
```

Include <sys/wait.h> and <stdlib.h> for definition of wait().
The execution of wait() could have two possible situations

1. If there are at least one child processes running when the call to wait() is made, the caller will be blocked until one of its child processes exits. At that moment, the caller resumes its execution

2. If there is no child process running when the call to wait() is made, then this wait() has no effect at all. That is, it is as if no wait() is there.

### EXAMPLE

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

void   main ( void ) {
        int status = 0;
        pid_t   pid = fork();
        if (pid  == 0){ // This check  will pass only for child  process
                printf("I am Child   process with pid %d  and i  am not
                       waiting\n",getpid());
                exit(status);
        }

        else if  (pid > 0){   // THIS Check  will pass only   for parent
                printf("I am Parent process with pid %d  and
                       i am waiting\n",getpid());
                pid_t exitedChildId = wait (&status);
                printf("I am Parent process and the child with pid %d
                       is exited\n",exitedChildId);
        }

        else if  (pid < 0){   // if fork() fails
                printf("Error in Fork");
        }
```

# Tasks

# 1. All tasks practice given in manual.

- **For Loop**
- **Switch**

2. Write a shell script using while loop that will display such pyramid of integer value.

```
0
1 0
2 1 0
3 2 1 0
4 3 2 1 0
5 4 3 2 1 0
6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
```

**3.Write a c++ program that print 1st 10 numbers using fork system call 1st 5 number print in child process 2nd 5 numbers print in parent process using fork and wait system call.also check the getpid and getppid of parent and child**

**4.Write a program even number print in parent and odd numbers in child process assume input from user.**