1. **Processing Environment**

1.4 Write the program to use wait/waitpid system call and explain what it do when call in parent

**Objectives:**

1. To learn about Processing Environment.

2. To know the difference between fork/vfork and various execs variations.

3. Use of system call to write effective programs.

**Theory:**

**Declarations:**

#include <sys/types.h>

#include <sys/wait.h>

pid\_t wait(int \*statloc);

/\* returns process ID if OK, or -1 on error \*/

pid\_t waitpid(pid\_t pid, int \*statloc, int options);

/\* returns process ID : if OK,

\* 0 : if non-blocking option && no zombies around

\* -1 : on error \*/

|  |  |
| --- | --- |
| wait() | waitpid() |
| wait blocks the caller until a child process terminates | waitpid can be either blocking or non- blocking:   * If options is 0, then it is blocking * If options is WNOHANG, then is it non-blocking |
| if more than one child is running then wait() returns the first time one of the parent's offspring exits | waitpid is more flexible:   * If pid == -1, it waits for any child process. In this respect, waitpid is equivalent to wait * If pid > 0, it waits for the child whose process ID equals pid * If pid == 0, it waits for any child whose process group ID equals that of the calling process * If pid < -1, it waits for any child whose process group ID equals that absolute value of pid |

Table 1.4 Data Dictonary

**Flowchart:**

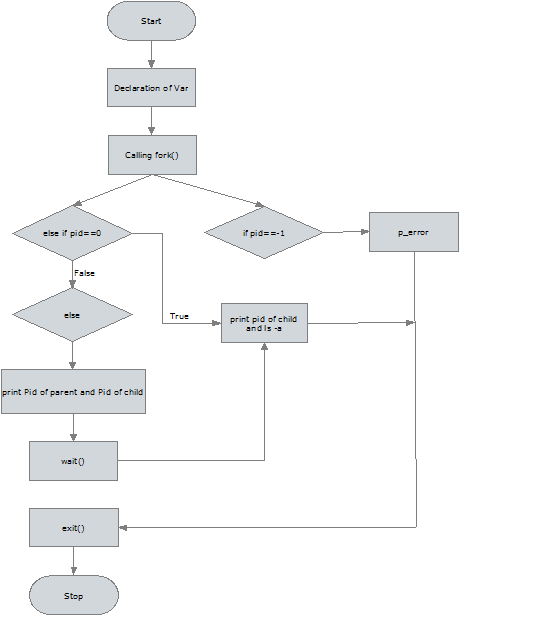


Fig 1.4 wait system call

**Data Dictionary:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr Number | Variable/Function | Datatype | Use |
| 1 | id | pid\_t | Process ID |
| 2 | i | int | Iterating for loop. |

Table 1.4 Data Dictonary

**Program:**

#include<stdlib.h>

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

void main()

{

pid\_t id=fork();

if(id==0)

{

printf("Child Process Started..ProcessID = %d\n", getpid());

printf("In Child\n");

for(int i=0;i<5;i++)

{

printf("In Child : %d\n",i);

}

printf("Child Finished\n");

exit(0);

}

else

{

printf("Parent Process Started..ProcessID = %d\n", getpid());

printf("In Parent\n");

printf("Parent waiting\n");

wait(NULL);

printf("Parent Resumed\n");

for(int i=0;i<5;i++)

{

printf("In parent : %d\n",i);

}

printf("Parent Finished\n");

}

}

**Output:**

sarita@sarita-HP-Laptop-15g-dr0xxx:~/Documents/TY/TY SEM II/UOS/2021uos/1/1d$ gedit wait.c

sarita@sarita-HP-Laptop-15g-dr0xxx:~/Documents/TY/TY SEM II/UOS/2021uos/1/1d$ gcc wait.c

sarita@sarita-HP-Laptop-15g-dr0xxx:~/Documents/TY/TY SEM II/UOS/2021uos/1/1d$ ./a.out

Parent Process Started..ProcessID = 1134565

In Parent

Parent waiting

Child Process Started..ProcessID = 1134566

In Child

In Child : 0

In Child : 1

In Child : 2

In Child : 3

In Child : 4

Child Finished

Parent Resumed

In parent : 0

In parent : 1

In parent : 2

In parent : 3

In parent : 4

Parent Finished

**Conclusion:**

1. The waitpid() call is more flexible than wait() system call as wait() would block.

2. The parent until child processes complete, while waitpid() can be implemented in blocking or unblocking ways.

**References:**

[1]www.tutorialspoint.com/unix\_system\_calls/