BiT
Bahir Dar University
Bahir Dar Institute of Technology

Department of Software Engineering

Faculty of Computing

Operating Systems and System Programming

System call Individual Assignment

Prepared By: Ermias Sintayehu Deresse

Id No. -BDU1307028

Submitted to: Instructor Wendimu Baye

Submission Date: 17- 11 - 2014

Table of Contents

1. What / Why / How, this system call?

The correct syntax to implement the function is:

```
#include <sys/wait.h>
```

```
  int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options)
```

The *waitid*() function shall suspend the calling thread until one child of the process containing the calling thread changes state. It records the current state of a child in the structure pointed to by *infop*. If a child process changed state prior to the call to *waitid*(), *waitid*() shall return immediately. If more than one thread is suspended in *wait()* or *waitpid()* waiting for termination of the same process, exactly one thread shall return the process status at the time of the target process termination.

The *waitid*() function shall suspend the calling thread until one child of the process containing the calling thread changes state. It records the current state of a child in the structure pointed to by *infop*. If a child process changed state prior to the call to *waitid*(), *waitid*() shall return immediately. If more than one thread is suspended in *wait()* or *waitpid()* waiting for termination of the same process, exactly one thread shall return the process status at the time of the target process termination.

The **waitid**() system call (available since Linux 2.6.9) provides more precise control over which child state changes to wait for.

The *idtype* and *id* arguments select the child(ren) to wait for, as follows:

- *idtype* == **P_PID**

Wait for the child whose process ID matches *id*.

- *idtype* == **P_PGID**

Wait for any child whose process group ID matches *id*.

- *idtype* == **P_ALL**

Wait for any child; *id* is ignored.

The child state changes to wait for are specified by one or more flags in the option.

- If WNOHANG was specified in *options* and there were no children in a waitable state, then waitid() returns 0 immediately and the state of the *siginfo_t* structure pointed to by *infop* depends on the implementation. To (portably) distinguish this case from that where a child was in a waitable state, zero out the *si_pid* field before the call and check for a nonzero value in this field after the call returns.

2. Briefly Discuss about list of parameter and flags

The parameter are listed in the following

- Idtype
- Id
- *infop
- Options

❖ Idtype : Specifies the child process.The idtype and id arguments are used to specify which children waitid() waits for.

⇒ If *idtype* is P_PID, *waitid*() shall wait for the child with a process ID equal to (**pid_t**)*id*.
⇒ If *idtype* is P_PGID, *waitid*() shall wait for any child with a process group ID equal to (**pid_t**)*id*.
⇒ If *idtype* is P_ALL, *waitid*() shall wait for any children and *id* is ignored

❖ Id: specifies the child process.
❖ Infop : specifies the location of a siginfo_t structure utilization information to be filled in with resource.
❖ options Specifies which state changes the waitid subroutine will wait for. The required memory flags are stated below:

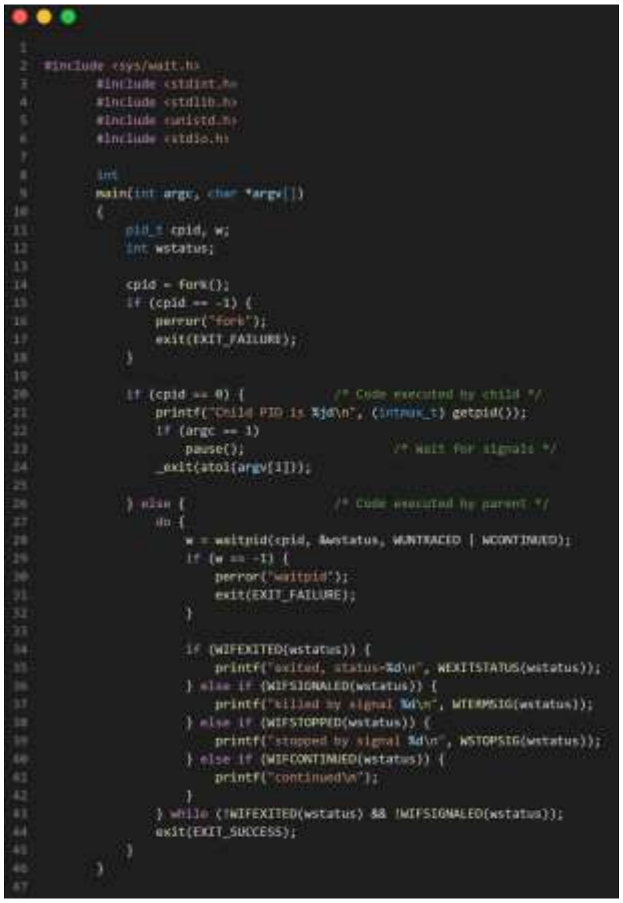
- WEXITED
  Wait for processes that have exited.
- WSTOPPED
  Status will be returned for any child that has stopped upon receipt of a signal.
- WCONTINUED
  Status will be returned for any child that was stopped and has been continued.
- WNOHANG
  Return immediately if there are no children to wait for.

- WNOWAIT
  Keep the process whose status is returned in the *infop* parameter in a waitable tate.This will not affect the state of the process. The process can be waited for again after this call completes.

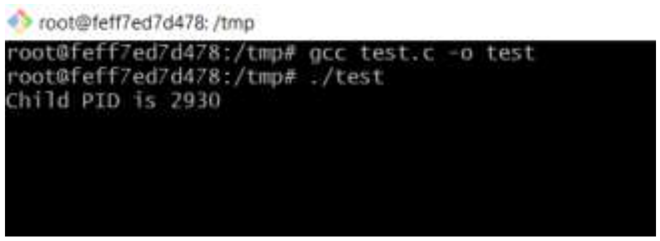
3. List the flags, their purpose with code implementation (give Example source code with output)

The flags are WEXITED,WSTOPPED,WCONTINUED,WNOHANG and WNOWAIT. They specifies in which the waited subroutine will wait for.The detailed description of flags is mentioned in the above question.

The following program demonstrates the use of fork() and waitid().The program creates a child process. If no command- line argument is supplied to the program, then the child suspends its execution using pause(), to allow the user to send signals to the child. Otherwise, if a command-line argument is supplied,then the child exits immediately, using the integer supplied on the command line as the exit status. The parent process executes a loop that monitors the child using waitid(), and uses the W*()macros described above to analyze the wait status value.

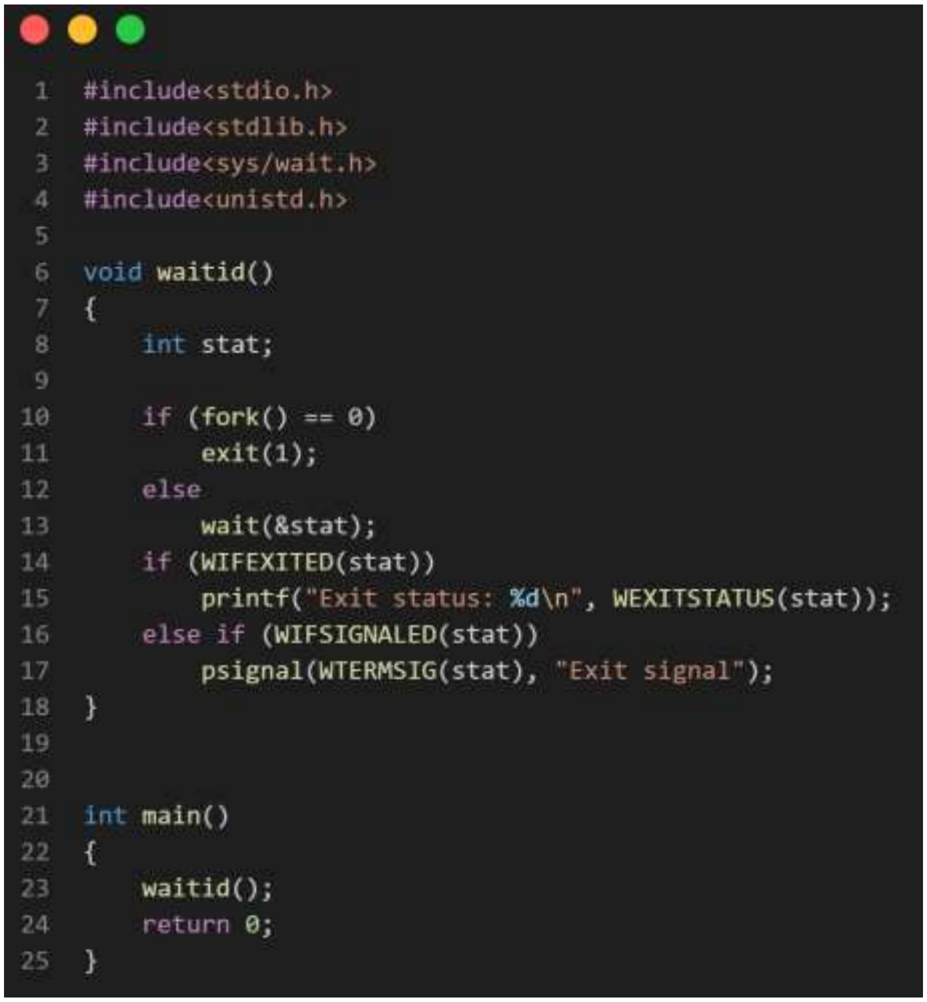


```

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

        int
        main(int argc, char *argv[])
        {
            pid_t cpid, w;
            int wstatus;

            cpid = fork();
            if (cpid == -1) {
                perror("fork");
                exit(EXIT_FAILURE);
            }

            if (cpid == 0) {            /* Code executed by child */
                printf("Child PID is %jd\n", (intmax_t) getpid());
                if (argc == 1)
                    pause();                    /* Wait for signals */
                _exit(atoi(argv[1]));

            } else {                    /* Code executed by parent */
                do {
                    w = waitpid(cpid, &wstatus, WUNTRACED | WCONTINUED);
                    if (w == -1) {
                        perror("waitpid");
                        exit(EXIT_FAILURE);
                    }

                    if (WIFEXITED(wstatus)) {
                        printf("exited, status=%d\n", WEXITSTATUS(wstatus));
                    } else if (WIFSIGNALED(wstatus)) {
                        printf("killed by signal %d\n", WTERMSIG(wstatus));
                    } else if (WIFSTOPPED(wstatus)) {
                        printf("stopped by signal %d\n", WSTOPSIG(wstatus));
                    } else if (WIFCONTINUED(wstatus)) {
                        printf("continued\n");
                    }
                } while (!WIFEXITED(wstatus) && !WIFSIGNALED(wstatus));
                exit(EXIT_SUCCESS);
            }
        }

```

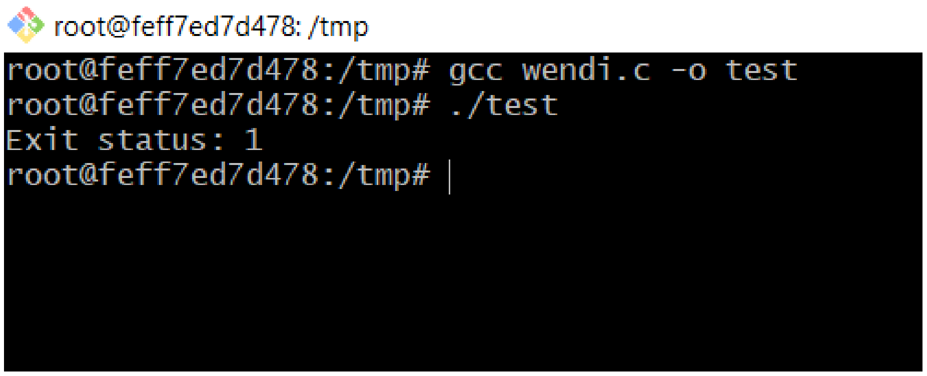


```
root@feff7ed7d478: /tmp
root@feff7ed7d478:/tmp# gcc test.c -o test
root@feff7ed7d478:/tmp# ./test
Child PID is 2930
```

- The following is about finding the exit status with WIFEXITED Flags.



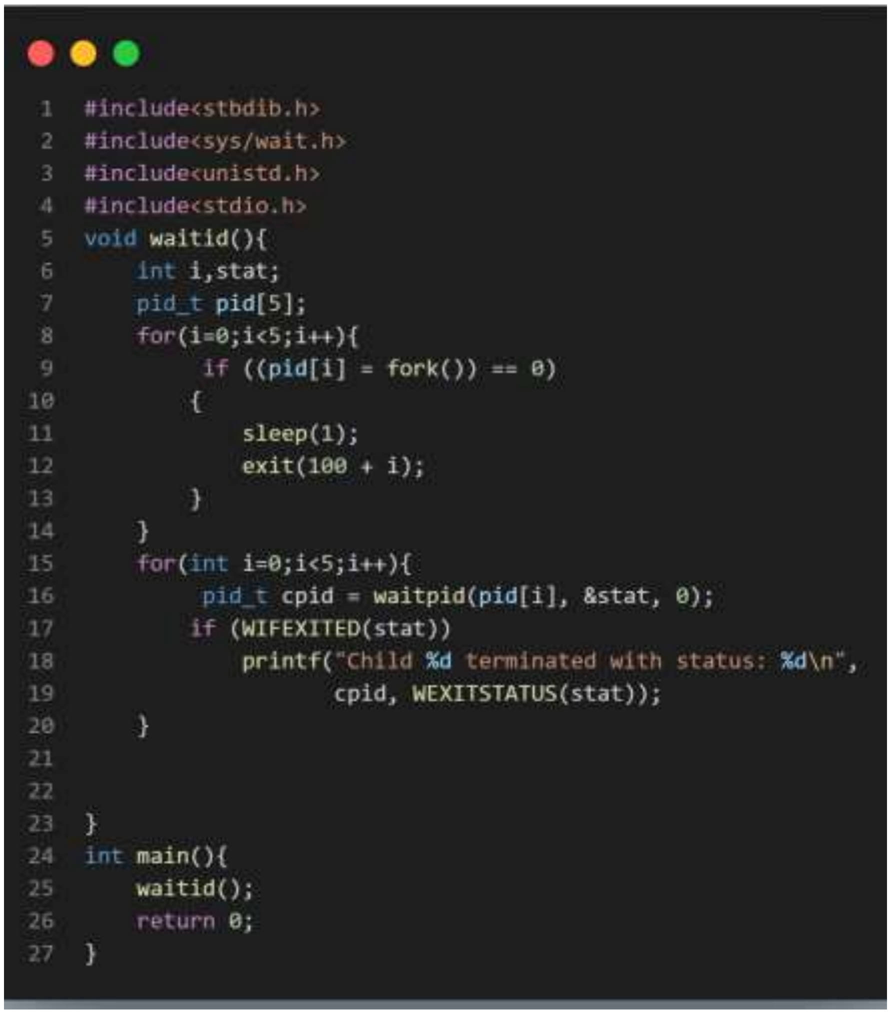
```c
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>

void waitid()
{
    int stat;

    if (fork() == 0)
        exit(1);
    else
        wait(&stat);
    if (WIFEXITED(stat))
        printf("Exit status: %d\n", WEXITSTATUS(stat));
    else if (WIFSIGNALED(stat))
        psignal(WTERMSIG(stat), "Exit signal");
}


int main()
{
    waitid();
    return 0;
}
```



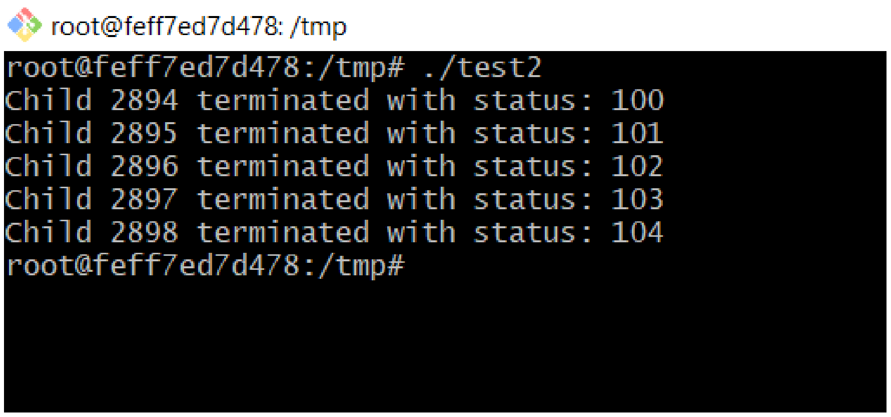
root@feff7ed7d478: /tmp

```
root@feff7ed7d478:/tmp# gcc wendi.c -o test
root@feff7ed7d478:/tmp# ./test
Exit status: 1
root@feff7ed7d478:/tmp# 
```

- The following code are also with implementations of WEXITSTATUS waitid flag in order to find the terminated status of child(ren).



```c
#include<stbdib.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdio.h>
void waitid(){
    int i,stat;
    pid_t pid[5];
    for(i=0;i<5;i++){
         if ((pid[i] = fork()) == 0)
        {
            sleep(1);
            exit(100 + i);
        }
    }
    for(int i=0;i<5;i++){
         pid_t cpid = waitpid(pid[i], &stat, 0);
        if (WIFEXITED(stat))
            printf("Child %d terminated with status: %d\n",
                   cpid, WEXITSTATUS(stat));
    }


}
int main(){
    waitid();
    return 0;
}
```



root@feff7ed7d478: /tmp

```
root@feff7ed7d478:/tmp# ./test2
Child 2894 terminated with status: 100
Child 2895 terminated with status: 101
Child 2896 terminated with status: 102
Child 2897 terminated with status: 103
Child 2898 terminated with status: 104
root@feff7ed7d478:/tmp#
```