Week6

1. Write a C program to demonstrate race condition between parent and child processes
2. Write a C program to demonstrate zombie status of a process and provide the solution for the same.
3. Write a C program to demonstrate the working of wait and waitpid functions

Solution:

1. Write a C program to demonstrate race condition between parent and child processes

```
#include "apue.h"

static void charatatime(char *);

int
main(void)
{
    pid_t   pid;

    if ((pid = fork()) < 0) {
        err_sys("fork error");
    } else if (pid == 0) {
        charatatime("output from child\n");
    } else {
        charatatime("output from parent\n");
    }
    exit(0);
}

static void
charatatime(char *str)
{
    char    *ptr;
    int     c;

    setbuf(stdout, NULL);           /* set unbuffered */
    for (ptr = str; (c = *ptr++) != 0; )
        putc(c, stdout);
}
```

Modified program to avoid race condition

```
   #include "apue.h"

   static void charatatime(char *);

   int
   main(void)
   {
       pid_t   pid;

+      TELL_WAIT();
+
       if ((pid = fork()) < 0) {
           err_sys("fork error");
       } else if (pid == 0) {
+          WAIT_PARENT();      /* parent goes first */
           charatatime("output from child\n");
       } else {
           charatatime("output from parent\n");
+          TELL_CHILD(pid);
       }
       exit(0);
   }

   static void
   charatatime(char *str)
   {
       char    *ptr;
       int     c;

       setbuf(stdout, NULL);           /* set unbuffered */
       for (ptr = str; (c = *ptr++) != 0; )
           putc(c, stdout);
   }
```

2. Write a C program to demonstrate zombie status of a process and provide the solution for the same.

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

int main()
{
   // fork() creates child process identical to parent
   int pid = fork();
```

```
    // if pid is greater than 0 than it is parent process
    // if pid is 0 then it is child process
    // if pid is -ve , it means fork() failed to create child process

    // Parent process
    if (pid > 0)
        sleep(20);

    // Child process
    else {
        exit(0);
    }

    return 0;
}
```

Using **wait()** system call: When the parent process calls wait(), after the creation of a child, it indicates that, it will wait for the child to complete and it will reap the exit status of the child. The parent process is suspended(waits in a waiting queue) until the child is terminated. It must be understood that during this period, the parent process does nothing just wait.

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

int main()
{
    int i;
    int pid = fork();
    if (pid==0)
    {
        for (i=0; i<20; i++)
```

```
        printf("I am Child\n");
    }
    else
    {
        wait(NULL);
        printf("I am Parent\n");
        while(1);
    }
}
```

C program to avoid zombie status of a process.

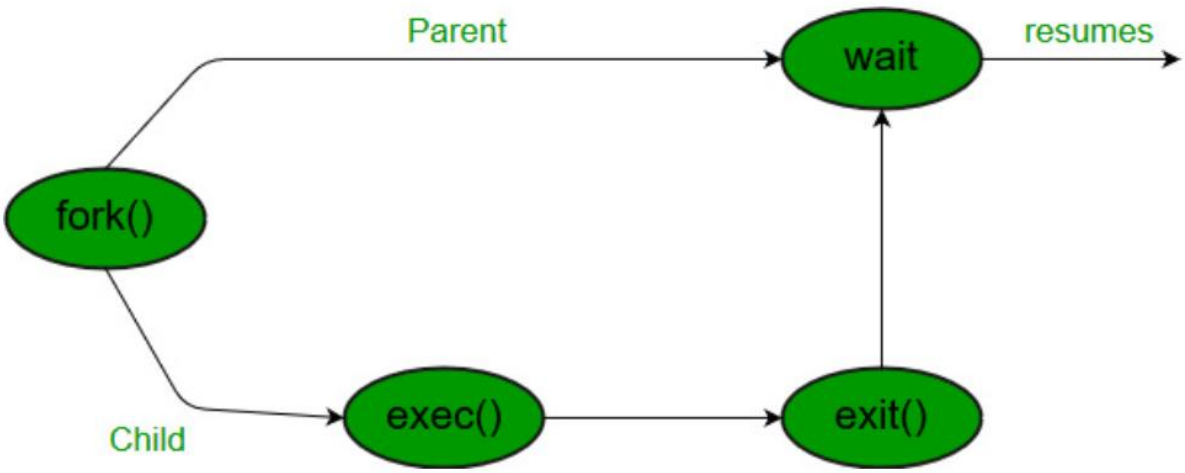
```
#include<stdio.h>
#include<stdlib.h>
#include <sys/wait.h>
int
main(void)
{
pid_t pid;
if ((pid = fork()) < 0) {
err_sys("fork error");
}
else if (pid == 0) { /* first child */
if ((pid = fork()) < 0)
        err_sys("fork error");
else if (pid > 0)
        exit(0);
sleep(2);
printf("second child, parent pid = %ld\n", (long)getppid());
exit(0);
}
if (waitpid(pid, NULL, 0) != pid)
        err_sys("waitpid error");
exit(0);
}
```

3. Write a C program to demonstrate the working of wait and waitpid functions

A call to **wait()** blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent continues its execution after wait system call instruction.

Child process may terminate due to any of these:

- It calls exit();
- It returns (an int) from main
- It receives a signal (from the OS or another process) whose default action is to terminate.



Syntax:

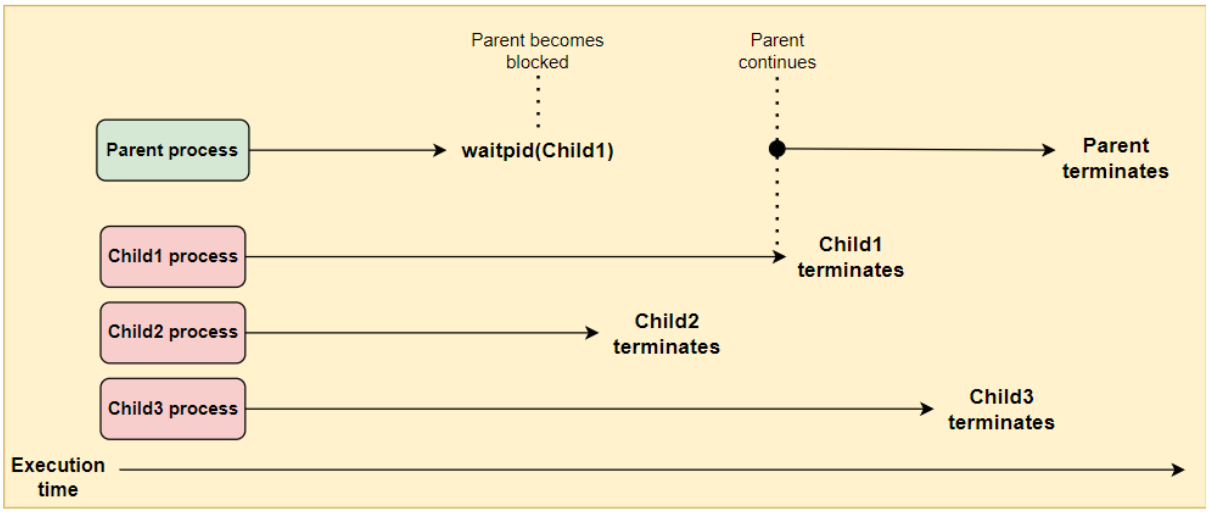
```
// take one argument status and returns
// a process ID of dead children.
pid_t wait(int *stat_loc);
```

```
// C program to demonstrate working of wait()
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
   pid_t cpid;
   if (fork()== 0)
      exit(0);          /* terminate child */
   else
      cpid = wait(NULL); /* bring in parent */
   printf("Parent pid = %d\n", getpid());
   printf("Child pid = %d\n", cpid);

   return 0;
}
```

- We know if more than one child processes are terminated, then **wait()** obtains any arbitrarily child process but if we want to obtain any specific child process, we use **waitpid()** function.
- **waitpid()** system call waits for a specific process to finish its execution. This system call can be accessed using our C programs' library sys/wait.h.



Syntax:

```
pid_t waitpid(pid_t pid, int *status_ptr, int options);
```

Let's discuss the three arguments that we provide to the system call.

- `pid`: Here, we provide the process ID of the process we want to wait for. If the provided `pid` is 0, it will wait for any arbitrary child to finish.
- `status_ptr`: This is an integer pointer used to access the child's exit value. If we want to ignore the exit value, we can use `NULL` here.
- `options`: Here, we can add additional flags to modify the function's behavior. The various flags are discussed below:
    - `WCONTINUED`: It is used to report the status of any child process that has been terminated and those that have resumed their execution after being stopped.
    - `WNOHANG`: It is used when we want to retrieve the status information immediately when the system call is executed. If the status information is not available, it returns an error.
    - `WUNTRACED`: It is used to report any child process that has stopped or terminated.'

- The system call will return the process ID of the child process that was terminated. If there is any error while waiting for the child process via the `waitpid()` system call, it will return `-1`, which corresponds to an error.

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

int main(){
   int cpid = fork();
   int cpid2 = fork();

   if(cpid != 0 && cpid2 != 0){
      int waitPID = 0;
      int status;
      printf("\nParent: I am going to wait for the process with process ID: %d\n", cpid2);

      while(waitPID == 0){
         waitPID = waitpid(cpid2, &status, WNOHANG);
      }

      printf("\nParent: Waited for child, the return value of waitpid(): %d\n", waitPID);
```

```
        printf("\nParent: The exit code of terminated child: %d\n", WEXITSTATUS(status));
        exit(1);
    }
    else if(cpid == 0 && cpid2 != 0){
        printf("\nChild1: My process ID is: %d, and my exit code is 1\n", getpid());
        exit(1);
    }
    else if(cpid != 0 && cpid2 == 0){
        printf("\nChild2: My process ID is: %d, and my exit code is 2\n", getpid());
        exit(2);
    }
    else{
        printf("\nChild3: My process ID is: %d, and my exit code is 3\n", getpid());
        exit(3);
    }

    return 0;
}
```