CS 214 Recitation(Sec. 6)

Zuohui Fu
Ph.D. Department of Computer Science
Office hour: Mon 2pm-3pm
Email: zf87 AT cs dot rutgers dot edu
10/24/2017

Topics

- Solution of HW 4
- Wait
- Signals

Solition-4.0

```
#include<sys/type.h>
#include<direct.h>

char * base = "./";
DIR *thingy = opendir(base);
struct dirent *newfile;
char buf[1024] = {0};
while((newfile = readdir(thingy)) != NULL) { }
```

Solution-4.1

```
while((newfile = readdir(thingy)) != NULL)
{
if (newfile->d_type == DT_REG) { }

 else if (newfile->d_type == DT_DIR) { }

else return 0;
```

//know how to use d_type--Format of a Directory Entry

Solition-4.2

- open function in <fcntl.h>:
  http://pubs.opengroup.org/onlinepubs/009695399/functions/open.html
- lseek function in <fcntl.h>:
  http://pubs.opengroup.org/onlinepubs/009695399/functions/lseek.html

Solition-4.2

```
1   //print the file name, full path and size
2   printf("s", buf);
3   int checkFD = open(buf,0_RDONLY);
4   //check if there is open error
5   if (checkFD == -1)
6   {
7       return -1;
8   }
9   int len = lseek(checkFD,0,SEEK_END);
10  close(checkFD);
11  printf("%d",len);
```

wait—system call()

- System call to *wait for a child*
  - pid_t wait (int *status)
- **wait()** are used to *wait for state changes* in a child of the calling process, and obtain information about the child whose state has changed. A state change is considered to be:
  - the child terminated;
  - the child was stopped by a signal;
  - the child was resumed by a signal.

wait()

- A process that calls ***wait()*** will:
  - **block the calling process**(if all of its children are *still running*)
  - **return immediately with the termination status of a child** (if a child has *terminated* and is waiting for its parent to accept its return code)
  - **return immediately with an error** (if it *doesn't have any child* processes)
- Return Value:
  - **wait()**: on success, returns the *process ID* of the terminated child; on error, -1 is returned

Zombie(defunct) processes

- A child process that terminates, but has not been waited for by its parent becomes a "**zombie**".
- A parent accepts a child's return code by executing **wait()**
- The kernel maintains a **minimal set of information** in the process table about the zombie process (PID, termination status, resource usage information, showing up as ***Z*** in ***ps -a***) in order to allow the parent to later perform a wait to obtain information about the child.

Why defunct process are created? How to find?

- When ever a process ends all the memory used by that process are cleared and assigned to new process but due to programming errors/bugs some processes are still left in process table. These are created when there is no proper communication between parent process and child process.

- ps -ef | grep defunct

- For more:
  https://amitvashist.wordpress.com/2015/01/01/defunct-processes/

Orphan processes

- A process whose parent process no more exists
- When a parent process is died, it is soon adopted by init process (*PID 1*)
- The parent process is either finished or terminated without waiting for its child process to terminate is called an **orphan process**
- It still **takes resources**, and having too many orphan process will overload init process. It is better not to have many orphan process.

Example-Zombie

```c
// A C program to demonstrate Zombie Process.
// Child becomes Zombie as parent is sleeping
// when child process exits.
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    // Fork returns process id
    // in parent process
    pid_t child_pid = fork();

    // Parent process
    if (child_pid > 0)
        sleep(50);

    // Child process
    else
        exit(0);

    return 0;
}
```

The child finishes its execution using exit() system call while the parent sleeps for 50 seconds, hence doesn't call wait() and the child process's entry still exists in the process table.

Example orphan

```c
// A C program to demonstrate Orphan Process.
// Parent process finishes execution while the
// child process is running. The child process
// becomes orphan.
#include<stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    // Create a child process
    int pid = fork();

    if (pid > 0)
        printf("in parent process");

    // Note that pid is 0 in child process
    // and negative if fork() fails
    else if (pid == 0)
    {
        sleep(30);
        printf("in child process");
    }

    return 0;
}
```

Parent finishes execution and exits while the child process is still executing and is called an orphan process now. However, the orphan process is soon adopted by init process, once its parent process dies.

Signals

- When the kernel recognizes an unexpected/unpredictable asynchronous events, it sends a signal to the process.
- Unexpected/unpredictable asynchronous events
  - floating point error
  - segmentation fault
  - control-C (termination request)
  - control-Z (suspend request)
  - Events are called interrupt s

Signals--conti

- What are signals for?
  - When a program forks into 2 or more processes, rarely do they execute independently.
  - The processes usually require some form of synchronization, often handled by signals.
- What are the two sources of signals?
  - Machine interrupts (such as typing the keyboard or hardward breakdown)
  - The program itself, other programs or the user (sent by system functions)

Signals--Example

```
// CPP program to illustrate
// default Signal Handler
#include<stdio.h>
#include<signal.h>

int main()
{
    signal(SIGINT, handle_sigint);
    while (1)
    {
        printf("hello world\n");
        sleep(1);
    }
    return 0;
}
```

Print hello world infinite times. If user presses ctrl-c to terminate the process because of SIGINT signal sent and its default handler to terminate the process.

For more:
http://www.thegeekstuff.com/2012/03/catch-signals-sample-c-code/

HW-Please also refer the attachment