C Language Concepts

CS 231

System Programming Topics

Process Control

- *Each process has a unique process ID*
- *getpid() returns ID of process*
- *getppid() returns ID of parent process*
- *getuid() returns real user ID of process (getgid for group)*
- *geteuid() returns effective process ID (getegid for group)*

Fork Function

- *Create a process by calling fork()*
  - *fork is called once, returns twice*
    - *once in the calling process*
    - *once in the created process*

Parent Process

- *A process calls fork() to create a child process*
- *If parent process dies before child, child is orphan and is adopted by init (process id 1)*
- *fork() returns value of 0 to child, child PID to parent*
- *Look at code forkPID.c*

Child and Parent are Nearly Identical

- *Child process inherits*
  - *real & effective user & group IDs*
  - *current working directory*
  - *root directory*
  - *open files*

Child Parent Differences

- *return value from fork()*
- *process ID*
- *parent process ID*
- *pending alarms cleared for child*
- *child pending signals is empty set*

Uses for fork()

- *Common for server program to wait for request, then fork() child to serve request, parent continues waiting for requests*
- *Process wants to execute different program: child calls exec to replace its code with new program.*

wait and waitpid functions

- *A parent waits on its child to get exit status*
- *If child dies and parent does not wait, child becomes a zombie process*
- *wait() blocks parent until a child dies*
- *waitpid() can wait for a specific child, or not block parent at all*

Wait Functions API

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

 pid_t wait(int *wstatus);

 pid_t waitpid(pid_t pid, int *wstatus, int options);

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

example code in parentWait.c

exec system calls

- *int execl(const char * path, const char * argo,...,const char* argn, char * /*NULL*/)*
- *int execv(const char * path, const char * argv[])*
- *int execle(const char * path, const char * argo,...,const char* argn, char * /*NULL*/, char * const envp[])*

exec calls continued

- *int execve(const char * path, const char * argv[], char * const envp[])*
- *int execlp(const char * file, const char * argo,...,const char* argn, char * /*NULL*/)*
- *int execvp(const char * file, const char * argv[])*

exec details

- *names are execl, execv, execle, execve, execlp, execvp*
- *l stands for list - arguments are character strings*
- *v stands for vector, array of pointers to arguments*
- *e stands for environment, array of pointers to environment strings*
- *p first argument is a file name, if '/' in name, it is a path, otherwise, PATH environment variable lists paths to search for file name*
- *see forkexecv.c and env.c*

IPC - Pipe

- *kernel data structure for interprocess communication*
- *information flows in one direction only*
- *processes must share a common ancestor*
- *there is a file descriptor for each pipe end*

pipe function

- *int pipe(int fileDes[2])*
- *fd[0] open for read*
- *fd[1] open for write*
- *call pipe in a process, then fork one or more children to establish communication*
- *close read end in writing process, and write end in reading process*

file descriptor

- *The operating system keeps track of all open files*
- *Everything the operating system can affect (in UNIX and variants) is a file*
- *Essentially, there is an array of open file information*
- *file descriptor is an index into array*
- *file descriptors may be duplicated with functions dup and dup2, see example code: pipe.c*

Signals

- *A signal is a message sent by OS to process*
- *The default response to most signals is for process to die*
- *The process can ignore (most) signals, or handle them with code*
- *See the code in signal1.c to signal4.c, illustrates use of signal function*
- *A more powerful function (gives programmer more control) is sigaction - it is also much more complicated*