CS 3411 Systems Programming

Department of Computer Science
Michigan Technological University

Unix Processes

Today's Topics

- Unix Processes
- Creating New Processes

Unix Processes

- Process is the image of a program in execution
- Processes are sequential in nature
- Processes may reside in memory simultaneously
- Time multiplex the CPU(s) to get required results

Creating Child Processes

- Use `fork()`!
- As usual, read the manual page.
- `fork()` creates a child process that is a copy of the parent process, with some exceptions
- Execution continues just after the `fork()` call in both processes
- `fork()` returns the PID of the new process to the parent, and 0 to the child process
- The child is a *copy* of the parent. *No memory is shared.*

fork()

```c
#include <unistd.h>

main() {
  fork();
  write(1, "Hi \n", 4);
}
```

```c
#include <unistd.h>

main() {
  if(fork() == 0) {
    /* Child writes: */
    write(1, "Hi \n", 4);
  }
  else {
    /* Parent writes: */
    write(1, "Hey \n", 5);
  }
}
```

fork()

```c
#include <unistd.h>
#include <stdio.h>

int x = 0;

main() {
  if(fork() == 0) {
    /* Child writes: */
    x++;
    printf("Child: x=%d\n", x); fflush(stdout);
  }
  else {
    /* Parent writes: */
    x++;
    printf("Parent: x=%d\n", x); fflush(stdout);
  }
}
```

fork()

- A forked child inherits open files of the parent
- The child process descriptor is a copy of the parent's process descriptor, except:
  - Return value from `fork()`
  - PID, PPID
  - Pending signals and alarms
  - File locks
  - Execution times

File Descriptors Example

```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

main() {
   int fd; char ch1, ch2;
   fd = open("datafile",O_RDWR);
   read(fd,&ch1,1);
   printf("In parent: ch1 = %c\n", ch1); fflush(stdout);
   if (fork() == 0) {
      /* Child */
      read(fd,&ch2,1);
      printf("In child: ch2 = %c\n", ch2); fflush(stdout);
   }
}
```

Executing a New Binary

- `execve() is used to execute a new program`
- Manual page!
- This function executes the program it is pointed to
- On success, `execve()` does not return: The process calling execve() is completely replaced by the newly executed process
- On error, -1 is returned
- File descriptors may be set to close on exec!

Creating a New Process

- Exec is most useful when used with fork
- In Unix, a new process is created by first forking an existing process, then calling a variant of exec from there
- Most process attributes are preserved, including the PID, PPID, file locks, pending signals, execution times and open files

execve() Example

```c
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

main() {
  char *a[4], *e[3];

  a[0] = "child";
  a[1] = "argument1";
  a[2] = "argument2";
  a[3] = NULL;

  e[0] = "ENV0=val0";
  e[1] = "ENV1=val1";
  e[2] = NULL;

  execve("child1", a, e);
  /* If we get here, something went wrong */
  perror("parent1");
  exit(1);
}
```

execve() Example

```
#include <stdio.h>

main(argc, argv, envp)
int argc;
char *argv[], *envp[];
{
   int i;
   char **ep;
   printf("child is running\n");
   for (i = 0; i < argc; i++) {
      printf("argv[%d]=%s\n",i,argv[i]);
   }
   for (ep = envp; *ep; ep++) {
      printf("%s\n", *ep);
   }
}
```

Convenience Calls To Exec

- `execl`, `execlp`, `execle`, `execv`, `execvp` are all convenience calls to `execve`
- The manual page has details!

More Examples

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
main(argc, argv)
int argc; char *argv[];
{
  int forkid, charnum;
  char fdval[20];
  if (argc != 3) {
    fprintf(stderr,"Usage: pexec filename charnum\n");
    exit(1);
  }
  if ((forkid = open(argv[1], O_RDONLY)) < 0 ) {
    fprintf(stderr, "Cannot open %s\n", argv[1]);
    exit(2);
  }
  sprintf(fdval, "%d", forkid); /* sprintf! */
  if (fork() == 0) {
    execl("pchild", "pchild", fdval, argv[2], (char *)0);
    fprintf(stderr, "Unable to exec\n");
    exit(3);
  }
  printf("Parent is after fork/exec\n");
}
```

More Examples

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

main(argc, argv)
int argc;char *argv[];
{
  int myfd;
  char gotch, val;
  if (argc != 3) {
    fprintf(stderr,"Usage: pchild filename charnum\n");
    exit(1);
  }
  myfd = atoi(argv[1]);
  gotch = atoi(argv[2]);
  lseek(myfd, (off_t)gotch, SEEK_SET);
  read(myfd, &val, 1);
  printf("Child got char %d from fd %d: %c\n", gotch, myfd, val);
}
```

More Examples

```
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
main(argc, argv)
int argc; char *argv[];
{
  int forkid, charnum;
  if (argc != 3) {
    fprintf(stderr,"Usage: pioexec filename charnum\n");
    exit(1);
  }
  if ((forkid = open(argv[1], O_RDONLY)) < 0 ) {
    fprintf(stderr, "Cannot open %s\n", argv[1]);
    exit(2);
  }
  if (fork() == 0) {
    close(0); dup(forkid); close(forkid);
    execl("piochild", "piochild", argv[2], (char *)0);
    fprintf(stderr, "Unable to exec\n");
    exit(3);
  }
  printf("Parent is after fork/exec\n");
}
```

More Examples

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

main(argc, argv)
int argc;char *argv[];
{
  int myfd;
  char gotch, val;
  if (argc != 2) {
    fprintf(stderr,"Usage: piokid charnum\n");
    exit(1);
  }
  gotch = atoi(argv[1]);
  lseek(0, (off_t)gotch, SEEK_SET);
  read(0, &val, 1);
  printf("Child got char %d from stdin: %c\n", gotch, val);
}
```