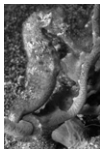
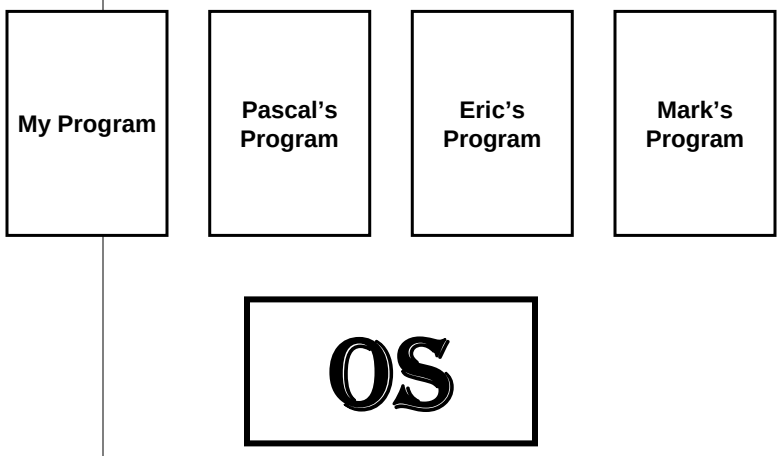
CS-034
C Meets Unix

Thomas Doeppner
Pascal Van Hentenryck



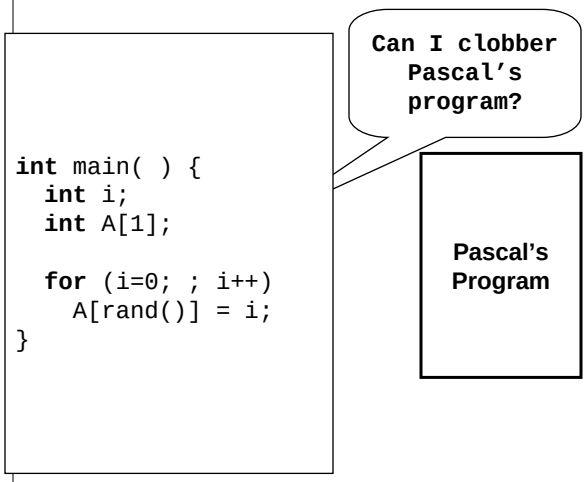
The Operating System



Processes

- **Containers for programs**
  - **virtual memory**
  - **scheduling**
  - **file references**
  - **and lots more!**

Idiot Proof …



Fair Share

```
long Ack(unsigned long i,
      unsigned long j){
  if (i == 0)
    return j+1;
  if (j == 0)
    return Ack(i-1, 1);
  return Ack(i-1,
      Ack(i, j-1));
}

int main( ) {
  Ack(4,4);
}
```

Can I monopolize the processor?

Pascal's Program

Creating Your Own Processes

```
#include <sys/types.h>
int main( ) {
  pid_t pid;
  if ((pid = fork()) == 0) {
     /* new process starts
        running here */
  }
  /* old process continues
     here */
}
```

Process IDs

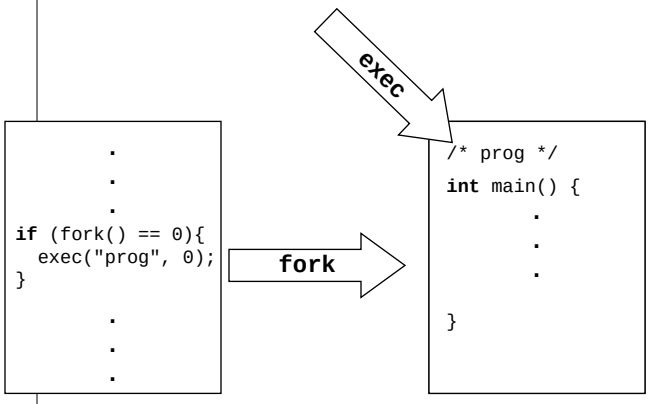
```
int main( ) {
  pid_t pid;
  pid_t ParentPid = getpid();

  if ((pid = fork()) == 0) {
     printf("%d, %d, %d\n",
          pid, ParentPid, getpid());
     exit(0);
  }
  printf("%d, %d, %d\n",
          pid, ParentPid, getpid());
}
```

```
parent prints:
  27355, 27342, 27342

child prints:
  0, 27342, 27355
```


Putting Programs into Processes



Exec

- **Family of related routines**
  - **we concentrate on one:**
    - **execl(program, arg0, arg1, … , 0)**

```
if (fork() == 0) {
  execl("./MyProg", "MyProg", 0);
}
```


A Random Program …

```
#define HowMany 17
int main() {
  int i;
  int stop = HowMany;

  for (i = 0; i < stop; i++)
    printf("%d\n", rand());
}
```

Passing It Arguments

- **From the shell**

```
% random 12
```

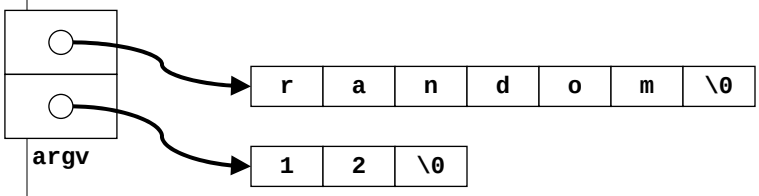
- **From a C program**

```
if (fork() == 0) {
  execl("./random", "random", "12", 0);
}
```


Receiving Arguments

```
int main(int argc, char *argv[]) {
  int i;
  int stop = atoi(argv[1]);

  for (i = 0; i < stop; i++)
    printf("%d\n", rand());
}
```



Not So Fast …

- **How does the shell invoke your program?**

```
if (fork() == 0) {
  execl("./random", "random", "12", 0);
}
/* what does it do here??? */
```


Wait

```
#include <sys/types.h>
#include <sys/wait.h>
…
  pid_t pid;
  int status;
  …
  if ((pid = fork()) == 0) {
    execl("./random", "random", "12", 0);
  }
  waitpid(pid, &status, 0);
```

Exit

```
int main( ) {
  pid_t pid;
  int status;
  if ((pid = fork()) == 0) {
    if (do_work() == 1)
      exit(0); /* success! */
    else
      exit(1); /* failure … */
  }
  waitpid(pid, &status, 0);
  /* low-order byte of status contains exit code */
  /* WEXITSTATUS(status) extracts it */
```


Whoops …

```
% SometimesUsefulProgram xyz
Are you sure you want to proceed? Y
Are you really sure? Y
Reformatting of your hard drive will begin
in 3 seconds.
Everything you own will be deleted.
There's little you can do about it.
Too bad …
```

One Approach …

A Gentler Approach

- **Signals**
  - **get a process's attention**
    - **send it a signal**
  - **process must either deal with it or be terminated**
    - **in some cases, the latter is the only option**

Sending a Signal

- `int kill(pid_t pid, int sig)`
  - **send signal *sig* to process *pid***
  - **(not always) terminate with extreme prejudice**
- **Also**
  - ***kill* shell command**
  - **type ctrl-c**
    - **sends signal 2 (SIGINT) to current process**
  - **do something illegal**
    - **bad address, bad arithmetic, etc.**

Handling Signals

```
#include <signal.h>

typedef void (*sighandler_t)(int);
sighandler_t signal(int signo,
    sighandler_t handler);

sighandler_t OldHandler;

OldHandler = signal(SIGINT, NewHandler);
```

Special Handlers

- **SIG_IGN**
  - **ignore the signal**
  - `signal(SIGINT, SIG_IGN);`
- **SIG_DFL**
  - **use the default handler**
    - **usually terminates the process**
  - `signal(SIGINT, SIG_DFL);`

Example

```
int main() {
  void handler(int);

  signal(SIGINT, handler);
  while(1)
    ;
  return 1;
}
void handler(int signo) {
  printf("I received signal %d. Whoopee!!\n", signo);
}
```

More Signals

- **SIGQUIT**
  - **produced from keyboard with ctrl-**
  - **default: terminates process and produces core dump**
- **SIGTERM**
  - **sent by shell's *kill* command if nothing else given**
  - **default: terminates process**
- **SIGKILL**
  - **default: terminates process**
  - **default cannot be overridden**

Even More Signals

- **SIGTSTP**
  - **produced from keyboard with ctrl-z**
  - **default: suspends process**
- **SIGCONT**
  - **sent by shell's *fg* (foreground) command**
  - **default: resumes process**
- **27 to 59 more, depending on the system**
  - **see "man 7 signal" for gory details**

Dealing with Failure

- ***fork*, *execl*, *wait*, *kill* directly invoke the operating system**
- **sometimes the OS says no**
  - **usually because you did something wrong**
  - **sometimes because the system has run out of resources**
  - **such "system calls" typically return -1 to signify a problem**

Reporting Failure

- **Integer error code placed in global variable *errno***
  - `extern int errno;`
  - **"man errno" lists all possible error codes and meanings**
  - **to print out most recent error**
    - `perror("message");`

Fork

```
int main( ) {
  while(1) {
    if (fork() == -1) {
      perror("fork");
      exit(1);
  }
}
```


Exec

```
int main( ) {
  if (fork() == 0) {
      execl("/garbage", "garbage", 0);
      /* if we get here, there was an
         error! */
      perror("exec");
      exit(1);
  }
}
```