UNIX Structure
The Unix Address Space

- stack
- dynamic
- bss
- data
- text
Creating a Process: Before

parent
process

fork( )
Creating a Process: After

parent process

fork( )
// returns p

child process

fork( )
// returns 0

(pid = p)
Fork and Wait

```c
short pid;
if ((pid = fork()) == 0) {
    /* some code is here for the child to execute */
    exit(n);
} else {
    int ReturnCode;
    while (pid != wait(&ReturnCode))
        ;
    /* the child has terminated with ReturnCode as its return code */
}
```
Process Control Blocks

<table>
<thead>
<tr>
<th>PID</th>
<th>Terminated children</th>
<th>Link</th>
<th>Return code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other Stuff</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process Control Block</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PID</th>
<th>Terminated children</th>
<th>Link</th>
<th>Return code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other Stuff</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PID</th>
<th>Return code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other Stuff</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PID</th>
<th>Terminated children</th>
<th>Link</th>
<th>Return code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other Stuff</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```c
int pid;
if ((pid = fork()) == 0) {
    /* we’ll soon discuss what might take place before exec is called */
    execl("/home/twd/bin/primes", "primes", "300", 0);
    exit(1);
}

/* parent continues here */

while(pid != wait(0)) /* ignore the return code */
    ;
```
Loading a New Image

Before

exec(prog, args)

After

prog’s text
prog’s data
prog’s bss
args
Quiz 1

```c
int A, B, C, D;
A=1;
if (fork() > 0) {
    B=1;
    A=111;
} else {
    C=2;
    if (fork() > 0) {
        D=222;
    } else {
        D=A+B+C;
        // what value is now in D for this process?
    }
}
exit(0);
```

Answer:

a) 0  
b) 3  
c) 113  
d) indeterminate
Representing the Address Space

• Important component of a process is its address space
  – how is it represented?

• Can page tables represent a process’s address space?
Simple User Address Space

- stack
- bss & dynamic
- data
- text
Address-Space Representation
Somewhat Simplified

- task_struct
- mm_struct
- vm_area_struct 0–7fff x, shared
- vm_area_struct 8000–1afff rw, private
- vm_area_struct 1b000–1bfff rw, private
- vm_area_struct 7fffd000–7fffffff rw, private
- struct file
Adding a Mapped File

- Stack
- Mapped file
- BSS & Dynamic
- Data
- Text
Address-Space Representation: More Areas

- task_struct
  - mm_struct
  - vm_area_struct 0–7fff x, shared
  - vm_area_struct 8000–1afff rw, private
  - vm_area_struct 1b000–1bfffe rw, private
  - vm_area_struct 200000–201fff rw, shared
  - vm_area_struct 7fffd000–7fffffff rw, private
  - struct file
Adding More Stuff

- stack 1
- stack 2
- stack 3
- mapped file 1
- mapped file 2
- mapped file 3
- mapped file 117
- bss & dynamic
- data
- text
Address-Space Representation: Reality

- task_struct
- mm_struct

200000–201fff

1b000–1bff

0–7fff

8000–1aff

202000–203fff

208000–210fff

204000–204fff

7fffd000–7fffffff
Layering

User Applications

System Calls

Virtual File System (VFS)

disk file systems (e.g. S5FS)
terminals
networking
other things (e.g. /proc)
Quiz 2

```c
int fd1 = open("file", O_CREAT|O_RDWR, 0666);
unlink("file");
write(fd1, "123", 3);
int fd2 = open("file", O_CREAT|O_RDWR, 0666);
write(fd2, "4", 1);
if (fork() == 0) {
    write(fd1, "5", 1);
}
exit(0);
```

The final contents of file are:

a) 12345  
b) 453  
c) 45  
d) 4
File-Descriptor Table

- File descriptor
- User address space
- Kernel address space
- File-descriptor table
  - ref count
  - access mode
  - file location
  - inode pointer
Allocation of File Descriptors

• Whenever a process requests a new file descriptor, the lowest numbered file descriptor not already associated with an open file is selected; thus

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

close(0);
fd = open("file", O_RDONLY);
```

— will always associate `file` with file descriptor 0 (assuming that the `open` succeeds)
Redirecting Output … Twice

```c
if (fork() == 0) {
    /* set up file descriptors 1 and 2 in the child process */
    close(1);
    close(2);
    if (open("/home/twd/Output", O_WRONLY) == -1) {
        exit(1);
    }
    if (open("/home/twd/Output", O_WRONLY) == -1) {
        exit(1);
    }
    execl("/home/twd/bin/program", "program", 0);
    exit(1);
}
/* parent continues here */
```
Redirected Output

File descriptor 1

File descriptor 2

User address space

Kernel address space

File-descriptor table

1 | WRONLY | 0 | inode pointer

1 | WRONLY | 0 | inode pointer
Redirected Output After Write

File descriptor 1
File descriptor 2
User address space

Kernel address space

File-descriptor table

```
1  WRONLY  100
```

```
1  WRONLY  0
```

inode pointer
Sharing Context Information

```c
if (fork() == 0) {
   /* set up file descriptors 1 and 2 in the child process */
   close(1);
   close(2);
   if (open("/home/twd/Output", O_WRONLY) == -1) {
      exit(1);
   }
   dup(1); /* set up file descriptor 2 as a duplicate of 1 */
   execl("/home/twd/bin/program", "program", 0);
   exit(1);
}
/* parent continues here */
```
Redirected Output After Dup

File descriptor 1

File descriptor 2

User address space

Kernel address space

File-descriptor table

2
WRONLY
100
inode pointer
Fork and File Descriptors

```c
int logfile = open("log", O_WRONLY);
if (fork() == 0) {
    /* child process computes something, then does: */
    write(logfile, LogEntry, strlen(LogEntry));
    ...
    exit(0);
}

/* parent process computes something, then does: */
write(logfile, LogEntry, strlen(LogEntry));
...```
File Descriptors After Fork

- **Parent's address space**
  - log file

- **Child's address space**
  - log file

- **Kernel address space**
  - File descriptor 2
  - Mode: WRONLY
  - Inode pointer
Directories

unix  etc  home  pro  dev

passwd  motd

twd

unix ...

slide1  slide2
## Directory Representation

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Inode Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>..</td>
<td>1</td>
</tr>
<tr>
<td>unix</td>
<td>117</td>
</tr>
<tr>
<td>etc</td>
<td>4</td>
</tr>
<tr>
<td>home</td>
<td>18</td>
</tr>
<tr>
<td>pro</td>
<td>36</td>
</tr>
<tr>
<td>dev</td>
<td>93</td>
</tr>
</tbody>
</table>
% ln /unix /etc/image
# link system call
## Directory Representation

<table>
<thead>
<tr>
<th>Directory</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>..</td>
<td>1</td>
</tr>
<tr>
<td>unix</td>
<td>117</td>
</tr>
<tr>
<td>etc</td>
<td>4</td>
</tr>
<tr>
<td>home</td>
<td>18</td>
</tr>
<tr>
<td>pro</td>
<td>36</td>
</tr>
<tr>
<td>dev</td>
<td>93</td>
</tr>
<tr>
<td>.</td>
<td>4</td>
</tr>
<tr>
<td>..</td>
<td>1</td>
</tr>
<tr>
<td>image</td>
<td>117</td>
</tr>
<tr>
<td>motd</td>
<td>33</td>
</tr>
</tbody>
</table>
Soft Links

% ln -s /unix /home/twd/mylink
% ln -s /home/twd /etc/twd
# symlink system call
Working Directory

• Maintained in kernel for each process
  – paths not starting from “/” start with the working directory
  – changed by use of the *chdir* system call
  – displayed (via shell) using “pwd”
    - how is this done?
Mount Points (1)
Mount Points (2)

- `/dev/dsk2` mounted as `/usr`

Diagram showing file systems:
- `/unix`
- `/etc`
- `/usr`
- `/mnt`
- `/dev`
- `/src`
- `/lib`
- `/bin`

tty01, tty02, dsk1, dsk2, tp1

mount /dev/dsk2 /usr
Representing File Systems

class fs {
    char dev[STR_MAX]; // device containing the f.s.
    char mountpt[STR_MAX]; // where the f.s. is mounted
    vnode *vnodecovered; // file on which f.s. is mounted
    vnode *root; // root of the f.s.
    virtual void read_vnode(vnode *);
    virtual void delete_vnode(vnode *);
};
Representing Files

```cpp
class vnode {
    unsigned short refcount;
    fs *vfsmounted;
    fs *vfs;
    unsigned long vno;
    int mode;
    int len;
    link_list_t link;
    kmutex_t mutex;
    virtual int create(const char *, int, vnode **);
    virtual int read(int, void *, int);
    virtual int write(int, const void *, int);
    ...
};
```
Mounting a File System (1)

/dev/disk0

/dev
mountpt
vnodecovered
root

vfs_root_vn

[/]

[/a/b]
Mounting a File System (2)

vfs_root_vn

/dev/disk0

/ 

[/]

/dev/disk1

/a/b

[/a/b]

[/a/b/…]
But Wait …

• What’s this about C++?

– real operating systems are written in C …
class fs {
  char dev[STR_MAX];
  char mountpt[STR_MAX];
  vnode *vnodecovered;
  vnode *root;
  virtual void read_vnode(vnode *);
  virtual void delete_vnode(vnode *);
};

typedef struct fs {
  char fs_dev[STR_MAX];
  char fs_mountpt[STR_MAX];
  struct vnode *fs_vnodecovered;
  struct vnode *fs_root;
  fs_ops_t *fs_op;
  /* function pointers */
  void *fs_i;
  /* extra stuff in subclasses */
} fs_t;
vnode

class vnode {
    unsigned short refcount;
    fs *vfsmounted;
    fs *vfs;
    unsigned long vno;
    int mode;
    int len;
    link_list_t link;
    kmutex_t mutex;
    virtual int create(const char *, int, vnode **);
    virtual int read(int, void *, int);
    virtual int write(int, const void *, int);
    ...
};

typedef struct vnode {
    unsigned short vn_refcount;
    struct fs *vn_vfsmounted;
    struct fs *vn_vfs;
    unsigned long vn_vno;
    int vn_mode;
    int vn_len;
    link_list_t vn_link;
    kmutex_t vn_mutex;
    struct vnode_ops *vn_op;
    /* function pointers */
    void *vn_i;
    /* extra stuff in subclasses */
}