COMP20200 Unix Programming

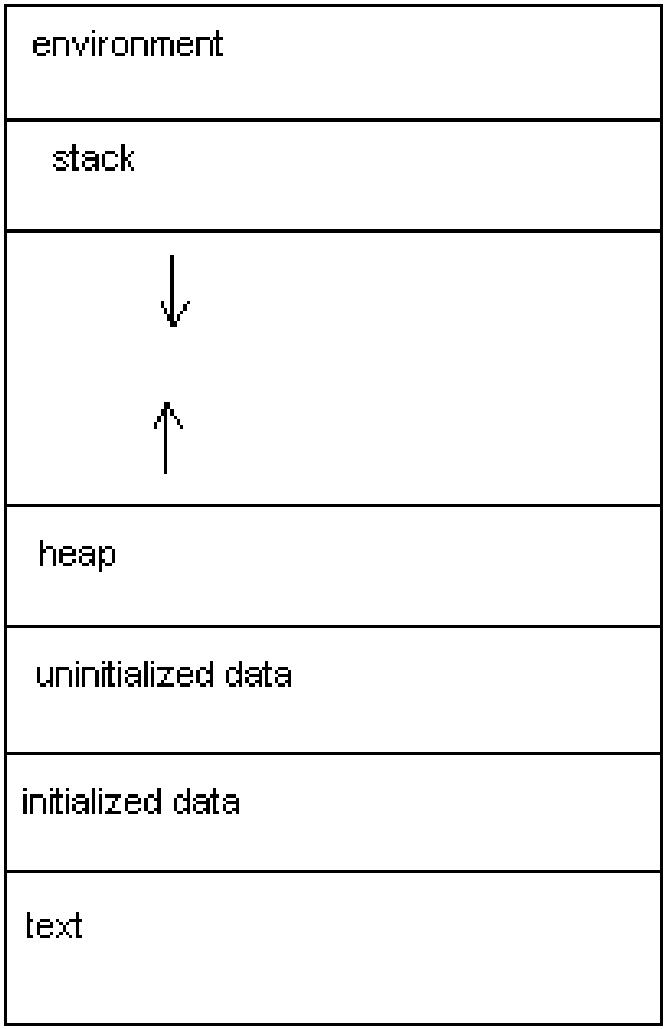
Lecture 7

CS, University College Dublin, Ireland

System Calls

- System calls expose functionality of OS to programmes
  - eg. write to hard disk, send & receive data on network.
- Normally program contained within virtual memory space.
  - System call allows program to break out.
- Many system calls, can be platform specific: code isn't portable.
- Standardised POSIX system calls

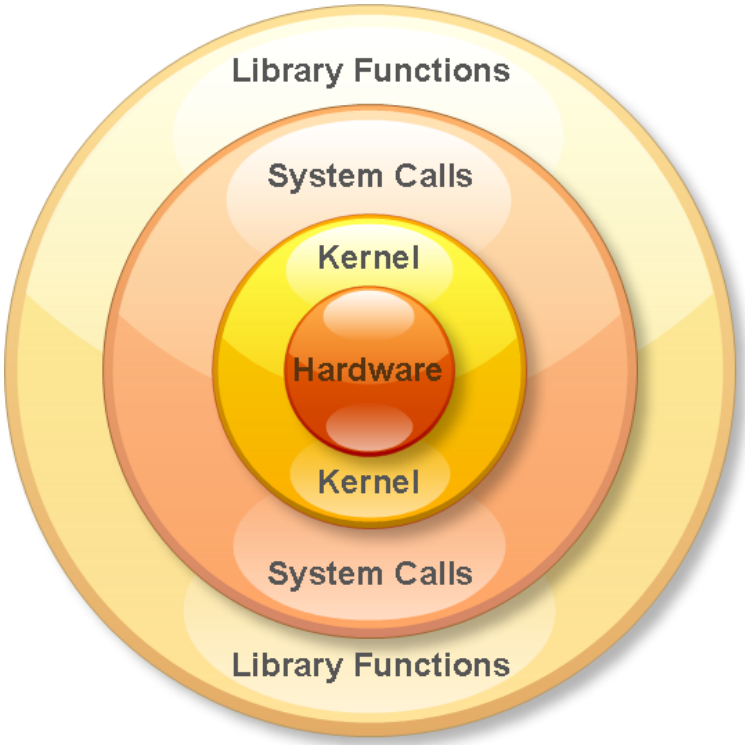
User Space

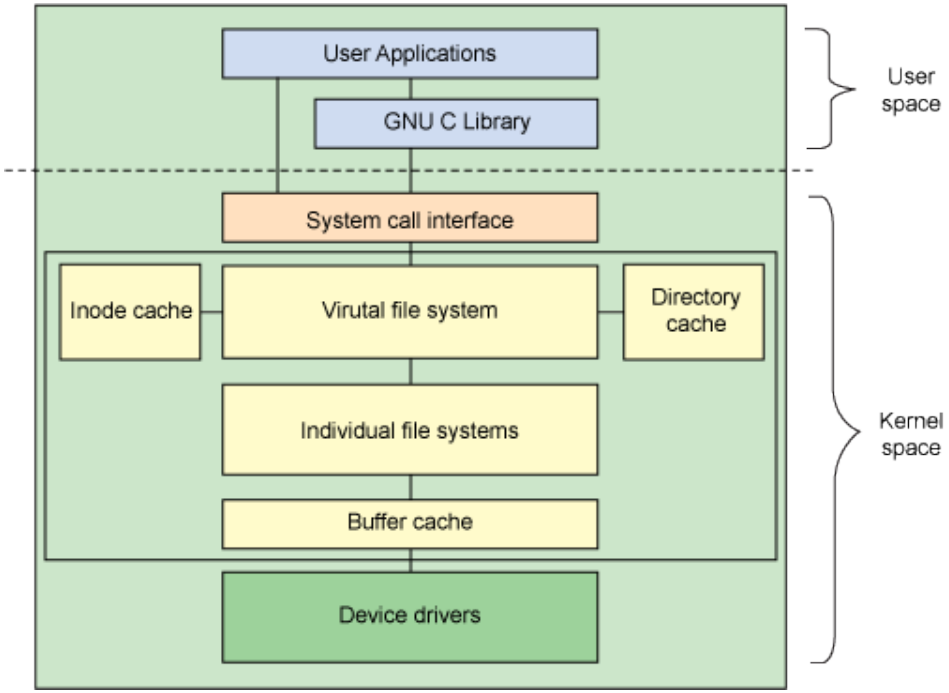


Virtual memory organization

System Calls

- System calls expose functionallity of OS to programmes





System Calls act on:

- Processes
- Files
- Networking Sockets
- Signals
- Inter-process communication
- terminals
- threads
- I/O devices

- System call invoked with a special instruction, written in assembly. `int 0x80`
- System-call wrappers in standard C library increase portability.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *pathname, int flags);)
```

- C function calls (eg. `fopen()`, `getchar()`, `printf()`) contain several system calls internally.

```
#include <stdio.h>
FILE *fopen(const char *path, const char *mode);
```

- System calls in 2nd Section of manual `man 2 intro; man 2 open`.
  - Library functions in 3rd Section of manual `man 3 intro; man 3 fopen`

Subset of system calls

- Managing I/O channels: `creat( )`, `open( )`, `close( )`
- Handling I/O operations: `read( )`, `write( )`
- For random access of files: `lseek( )`
- Aliasing and removing files: `link( )`, `unlink( )`
- Getting file status: `stat( )`
- Access control: `access( )`, `chmod( )`, `chown( )`
- Process control: `exec( )`, `fork( )`, `wait( )`, `exit( )`
- Process ownership: `getuid( )`
- Process ID: `getpid( )`
- Process control: `signal( )`, `kill( )`, `alarm( )`
- Changing working directory: `chdir( )`
- Manipulate low level memory attributes: `mmap()`, `shmget()`, `mprotect()`, `mlock()`
- Time management: `time()`, `gettimer()`, `settimer()`, `settimeofday()`, `alarm()`
- Creating inter-process communication: `pipe( )`

- System call:

```
#include <sys/mman.h>
void *mmap(void *addr, size_t length, int prot,
           int flags, int fd, off_t offset);
int munmap(void *addr, size_t length);
```

- C Library function

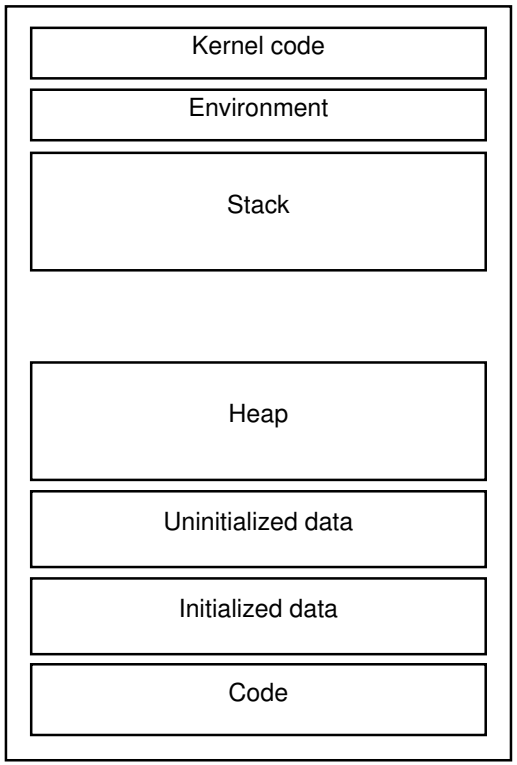
```
#include <stdlib.h>
void *malloc(size_t size);
void free(void *ptr);
```

- Almost always use function calls

fork( )

```
#include <unistd.h>
pid_t fork(void);
```

- Create a new process by duplicating the existing process.
- Existing process becomes the parent,
- newly created process becomes the child process.
- child has a unique PID
- Virtual memory duplicated with copy-on-write (COW)
  - Only when write to memory is physical memory copied.
- Code section also duplicated: same program.
  - Different action based on PID



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

int globalv;
int main(void){
  pid_t childPID;
  int localv = 0;
  childPID = fork();
  if(childPID >= 0) { // fork was successful
    if(childPID == 0) { // child process
      localv++; globalv++;
    printf("Child local:%d global:%d\n", localv, globalv);
    } else { //Parent process
      localv = 10; globalv = 20;
    printf("Parent local:%d global:%d\n", localv, globalv);
    }
  } else { // fork failed
    printf("\n Fork failed.\n");
    return 1; }
  return 0;
}
```

Simple copy with system calls

```
/* cp:  copy f1 to f2 */
...
  if (argc != 3)
    error("Usage: cp from to");
  if ((f1 = open(argv[1], O_RDONLY, 0)) == -1)
    error(...);
  if ((f2 = creat(argv[2], PERMS)) == -1)
    error(...);
  while ((n = read(f1, buf, BUFSIZ)) > 0)
    if (write(f2, buf, n) != n)
      error(...);
  close(f1);
  close(f2);
  ...
```

Note: Full code on next slide

```c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#define PERMS 0666      /* RW for owner, group, others */
void error(char *, ...);

/* cp:  copy f1 to f2 */
int main(int argc, char *argv[]) {
  int f1, f2, n;
  char buf[BUFSIZ];
  if (argc != 3)
    error("Usage: cp from to");
  if ((f1 = open(argv[1], O_RDONLY, 0)) == -1)
    error("cp: can't open %s", argv[1]);
  if ((f2 = creat(argv[2], PERMS)) == -1)
    error("cp: can't create %s, mode %03o",
        argv[2], PERMS);
  while ((n = read(f1, buf, BUFSIZ)) > 0)
    if (write(f2, buf, n) != n)
      error("cp: write error on file %s", argv[2]);
  close(f1); close(f2);
  return 0; }
```

- Command line tip(s) of the day:
  - See sorted list of running processes:
    - `top`
  - List all processes:
    - `ps aux`
  - Stop a process
    - `kill <pid>`
    - Force stop of unresponsive: `kill -9 <pid>`
  - Find/kill process by name:
    - `pgrep`
    - `pkill`
- Vi tip(s) of the day:
  - `:sp` Split screen, `:vsp` Vertical split screen
  - `Ctrl+w[jkhl]` or `Ctrl+w Ctrl+w` to move between screens
  - `:tabnew <file name>` Opens new tab.
  - `gt` or `gT` move between tabs
  - `qa` quit all, `wqa` write & quit all