OPERATING SYSTEMS: INTRODUCTION AND BASIC CONCEPTS

Operating System Services

Goals

- To understand what an operating system service is.
- To understand mechanisms participating in a system call.
- To know POSIX interface features.
- To know main services offered by POSIX.

Running the Operating System

- Once startup is finished, the operating system only runs in response to interrupts.

- Operating System activated in response to:
  - A service request from a process.
  - An interrupt (peripheral or clock).
  - Hardware exception.

Service activation

- A direct invocation to an OS routine leads to security problems:
  - How to perform changes in secure mode of operation?

- Using a software interrupt, OS activation becomes safer.
  - Library routine:
    - Machine instructions to prepare OS call.
    - Trap instruction.
    - Instructions processing results from OS call.

OS services: System calls.

- Interface between applications and OS.
  - Generally available as functions in assembler.
  - Currently also in high-level languages (C, C++, …).

- Typical services from OS.
  - Process management.
  - Thread management.
  - Signals and timers management.
  - Memory management.
  - Files and directories management.

- Example calls:
  - read: Allows reading data from file.
  - fork: Allows creating a new process.

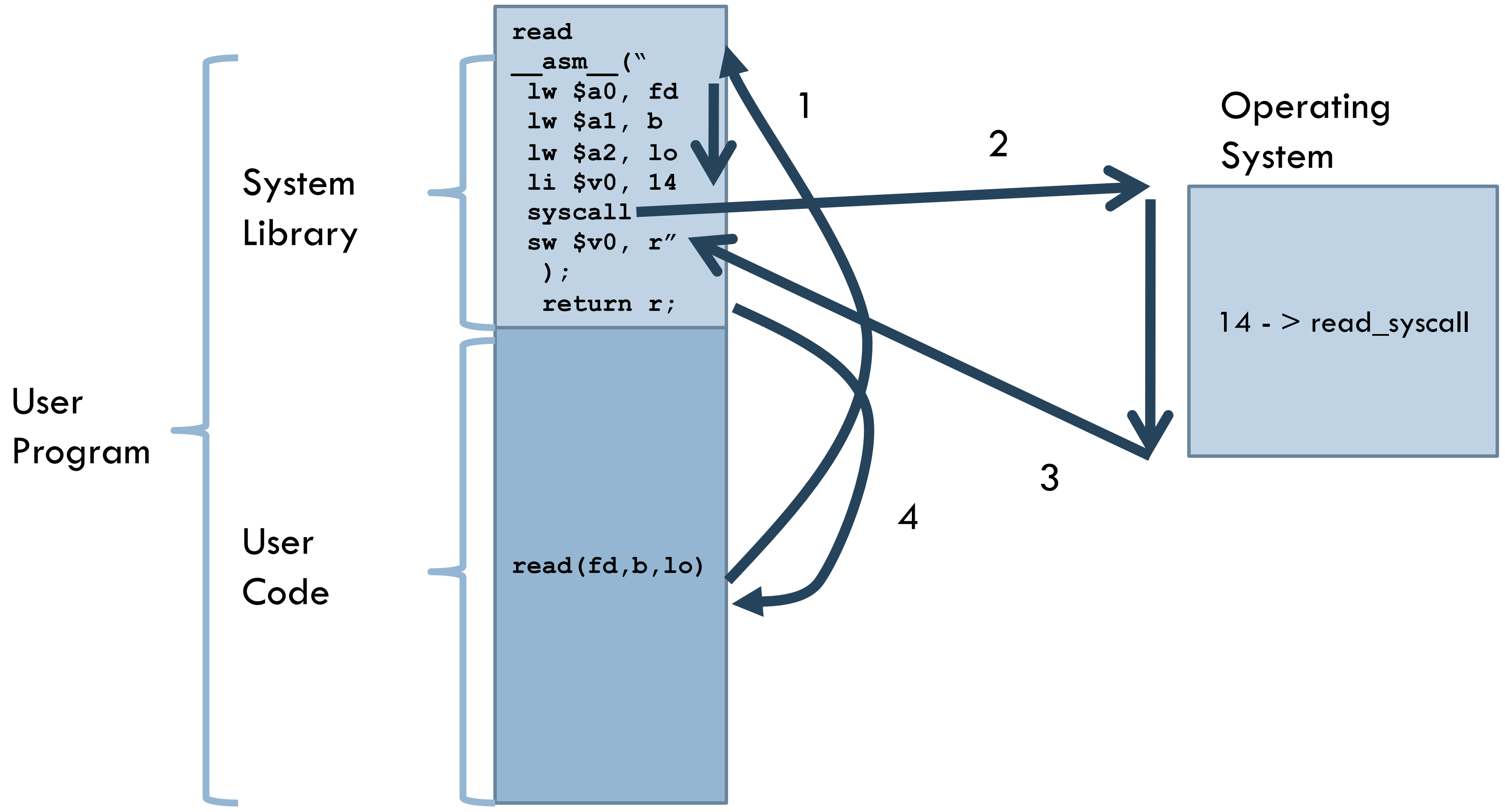
Call invocation

- Each **API** (*Applications Programmer Interface*) corresponds t some OS **service**.
  - Function is a wrapper for the code invoking the OS service.

- Includes the execution of a **trap** instruction to transfer control to operating system by generating an interrupt.

- Operating system handles interrupt and returns control to user program.

Call invocation



```
read
__asm__("
 lw $a0, fd
 lw $a1, b
 lw $a2, lo
 li $v0, 14
 syscall
 sw $v0, r"
  );
  return r;
```

```
read(fd,b,lo)
```

Service selection

- A single **trap** instruction and multiple services.
  - Need some mechanism for parameter passing from user process to kernel.

- As a minimum, a specification of desired service needs to be passed.
  - Usually a **numeric descriptor**.

Parameter passing

- Three generic methods to pass parameters for system calls:
  - In **registers**.
  - In a **table in memory**, which address is passed to OS through a register.
  - Placing parameters in **program stack** and allow OS to extract them.

- Each OS provides its own **system calls**:
  - POSIX in UNIX and LINUX.
  - Win32 in Windows NT.

Handling routine

- Handling routine must:
  - Retrieve parameters sent by the user process.
  - Identify service to be executed.
  - Determine address of service routine (indexing in table of service routines).
  - Transfer control to service routine.

Call invocation

```
int read(int fd, char * b, int lon) {
  int r;
  __asm__(“
    lw $a0, fd
    lw $a1, b
    lw $a2, lon
    li $v0, 14
    syscall
    sw $v0, r”
  );
  return r;
}
```

READ_SYSCALL

TRAP

Programmer interface

- Interface offers view as extended machine that operating system user has.
- Each operating system may offer one or several interfaces:
  - Linux: POSIX
  - Windows: Win32, POSIX

POSIX standard

- Standard interface for operating systems from IEEE.
- **Goal**: applications portability for different platforms and operating systems.
- **It is NOT** an implementation.
  - It only defines an interface.
- A family of standards

1003.1	OS basic services.
1003.1a	Extensions to basic services.
1003.1b	Real time extensions.
1003.1c	Threading extensions.
1003.2	Shell and utilities.
1003.2b	Additional utilities.

POSIX characteristics

- Short function names in lower case:
  - **fork**
  - **read**
  - **close**

- Functions usually return 0 on success or -1 on error.
  - Variable **errno**.

- Resources managed by operating system referenced through **descriptors** (integer numbers).

Example: Running a command

```
#include <sys/types.h>
#include <stdio.h>
int main(int argc, char** argv) {
  pid_t pid;
  pid = fork();
  switch (pid) {
    case -1: /* error */
      exit(-1);
    case 0: /* child process */
      if (execvp(argv[1], &argv[1])<0) { perror("error"); }
        break;
    default:
      printf("Parent process");
  }
  return 0;
}
```

```
prog cat f1
```

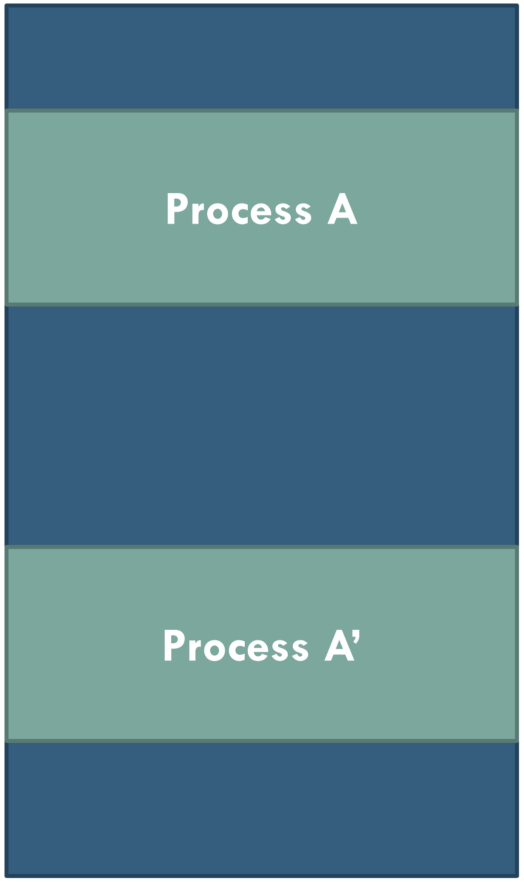
Fork service

- `pid_t fork(void);`

- Duplicates process invoking the call.
- Parent process and child process go on running the same program.
- Child process inherits open files from parent process.
  - Open file descriptors are copied.
- Pending alarms are deactivated.

- Returns:
  - -1 on error.
  - In parent process: child process descriptor.
  - In child process: 0.

Fork service



FORK

Exec service

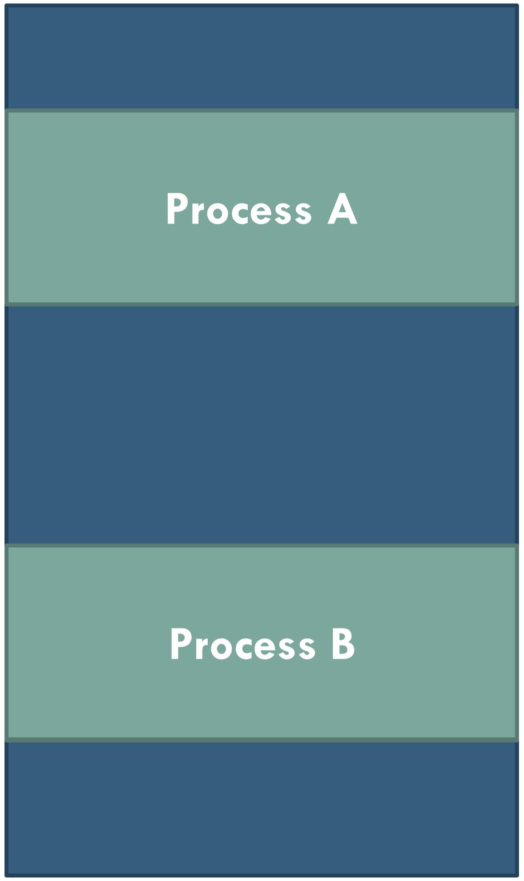
- Single service with multiple library functions.

```
int execl(const char *path, const char *arg, ...);
int execv(const char* path, char* const argv[]);
int execve(const char* path, char* const argv[], char* const envp[]);
int execvp(const char *file, char *const argv[])
```

- Changes current process image.
  - **path**: path to executable file.
  - **file**: Looks for the executable file in all directories specified by PATH.

- Description:
  - Returns -1 on error, otherwise it does not return.
  - The same process runs another program.
  - Open files remain open.
  - Signals with default action remain defaulted, signals with handler take default action.

Exec service



EXEC

Exit service

- Finalizes process execution.

```
void exit(status);
```

- All open files descriptors are closed.
- All process resources are released.
- **PCB** (Process Control Block) is released.

Example: Running a command

```
#include <sys/types.h>
#include <stdio.h>
int main() {
  pid_t pid;
  int status;

  pid = fork();
  if (pid == 0)    { /* child process */
    execlp("ls","ls","-l",NULL);
    exit(-1);
  }
  else            /* parent process */
    printf("Parent finalized\n");
  return 0; /* Invokes exit(0) */
}
```

Generic operations on files

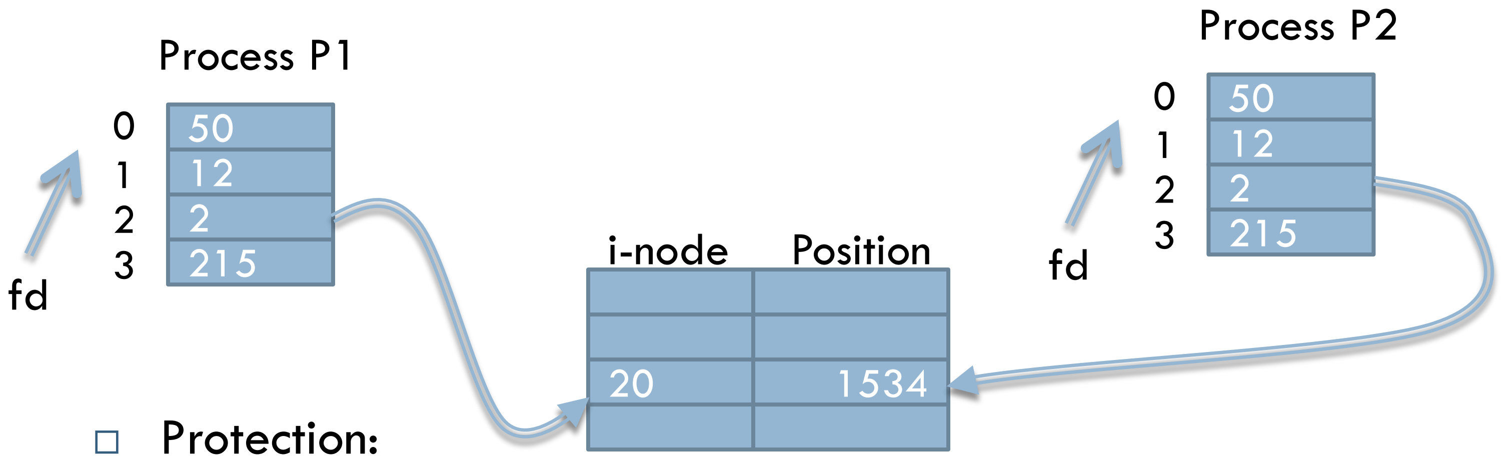
- **create**: Creates a file with name and attributes.
- **delete**: Deletes a file.
- **open**: Opens a file to allow access operations.
- **close**: Closes an open file.
- **read**: Reads data from open file to a memory buffer.
- **write**: Writes data to an open file from memory buffer.
- **position**: Moves pointer used to access file affecting subsequent operations.
- **control**: Allows manipulation of file attributes.

POSIX services for files

- **Logical view:**
  - A file.
- Keeps pointer associated to every open file.
  - **Pointer**: position where next operation starts.
- Most operations work with file descriptors:
  - An integer number form 0 to 64K
  - Obtained upon opening file.
  - Rest of operations use file descriptors.
- Predefined descriptors:
  - 0: standard input.
  - 1: standard output.
  - 2: error output.

POSIX services for files

- Every process has associated an open files table.
- When a process is duplicated (fork):
  - Duplicates open files table.
  - Shares intermediate table with i-nodes and positions.



- Protection:
  - `owner   group world`
  - `rwx     rwx   rwx`
- Examples: `755` is `rwxr-xr-x`

Files, directories and POSIX services

- File types:
  - Regular.
  - Directory.
  - Special.
- Names for files and directories:
  - Full name (starts with /)
    - `/usr/include/stdio.h`
  - Name relative to current directory (does not start with /)
    - `stdio.h` assuming `/usr/include` is current directory.
  - `.` and `..` entries can be used to form paths:
    - `../include/stdio.h`

CREAT – Create file

- Servicio:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int creat(char *name, mode_t mode);
```

- Arguments:
  - **`name`** File name
  - **`mode`** Access rights bits.
- Returns:
  - Return file descriptor or -1 upon error.

CREAT – create file

- Description:
  - File is open for writing.
  - If not existing, creates an empty file.
    - UID_owner = UID_effective
    - GID_owner = GID_effective
  - If existing, truncates without changing access rights bits.

- Examples:

```
fd = creat("data.txt", 0751);
fd = open("data.txt",
          O_WRONLY | O_CREAT | O_TRUNC, 0751);
```

UNLINK – Erase file

- Service:

```
#include <unistd.h>
int unlink(const char* path);
```

- Arguments:
  - **`path`** file name
- Returns:
  - Returns 0 or -1 upon error.
- Description:
  - Decrements link counter. If counter is 0, erases file and releases resources.

OPEN – Open a file

- Service:

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

- Arguments:
  - `name` file name.
  - `flags` options for opening:

`O_RDONLY`	Read only.
`O_WRONLY`	Write only.
`O_RDWR`	Read/write
`O_APPEND`	Access pointer moves to file end.
`O_CREAT`	If existing has no effect. If not existing creates.
`O_TRUNC`	Truncates if open for writing.

Open – Opening a file

- Returns:
  - A file descriptor or -1 upon error.

- Examples:

```
fd = open("/home/peter/data.txt");
fd = open("/home/peter/data.txt",
           O_WRONLY | O_CREAT | O_TRUNC, 0750);
```

CLOSE – Closing a file

- Service:

```
int close(int fd);
```

- Arguments:
  - `fd` file descriptor.
- Returns:
  - Zero or -1 upon error.
- Description:
  - Process looses its link with the file.

READ – Reading from a file

- Service:

```
#include <sys/types.h>
ssize_t read(int fd, void *buf, size_t n_bytes);
```

- Arguments:

`fd`	File descriptor.
`buf`	Buffer for data storage.
`n_bytes`	Number of bytes to be read

- Returns:
  - Number of bytes effectively read or -1 upo error.
- Description:
  - Transfers `n_bytes`.
  - Can read less bytes when end of file is reached or interrupted by a signal.
  - After reading the file pointer is incremented with the number of bytes effectively read.

WRITE – Writing to a file

- Service:

```
#include <sys/types.h>
ssize_t write(int fd, void *buf, size_t n_bytes);
```

- Arguments:

`fd`	File descritor.
`buf`	Buffer with data to be written.
`n_bytes`	Number of bytes to be written.

- Returns:
  - Number of bytes effectively written or -1 upon error.
- Description:
  - Transfers `n_bytes`.
  - It may write less data than requested in file maximum size is reached or interrupted by a signal
  - After writing file pointer is incremented with the number of bytes effectively written.
  - If end of file is rebased, the file size is increased.

LSEEK – Moving the file pointer

- Service:

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int fd, off_t offset, int whence);
```

- Arguments:

`fd`	File descriptor.
`offset`	Offset from base position.
`whence`	Base position for offset.

- Returns:
  - New position or -1 upon error.
- Description:
  - Repositions pointer associated to a `fd`
  - New position computation:
    - **SEEK_SET** position = offset
    - **SEEK_CUR** position = current position + offset
    - **SEEK_END** position = file size + offset

FNCTL – Attribute modification

- Service:

```
#include <sys/types.h>
int fnctl(int fildes, int cmd /* arg*/ ...);
```

- Arguments:

`fildes`	File descriptor
`cmd`	Command to modify attributes.

- Returns:
  - 0 on success or -1 upon error.
- Description:
  - Modifies attributes for an open file.

DUP – Duplicate a file descriptor

- Service:

```
int dup(int fd);
```

- Arguments:
  - `fd` file descriptor
- Returns:
  - A file descriptor sharing all the propoerties of **fd** or -1 upon error.
- Description:
  - Creates a new file descriptor having in common with the previous one:
    - Accesses to the same file.
    - Shares the same position pointer.
    - Access mode is identical.
  - New decriptir gets the lowest avaialbe numeric value.

FTRUNCATE – Space allocation for a file

- Service:

```
#include <unistd.h>
int ftruncate(int fd, off_t length);
```

- Arguments:

`fd`	File descriptor.
`length`	New file size.

- Returns:
  - Return 0 or -1 upon error.
- Description:
  - New file size is `length`.
  - If `length` es 0 file is truncated.

STAT – Information on a file

- Service:

```
#include <sys/types.h>
#include <sys/stat.h>
int stat(char *name, struct stat *buf);
int fstat(int fd, struct stat *buf);
```

- Arguments:
  - `name`	File name.
  - `fd`	File descriptor.
  - `buf`	Pointer to object of type `struct stat`
  - File information stored in buf.
- Returns:
  - 0 on success or -1 upon error.

STAT – Information on a file

- Description:
  - Gets information about a file and stores in object of type `struct stat`:

```
struct stat {
        mode_t  st_mode;  /* file mode*/
        ino_t   st_ino;   /* i-node */
        dev_t   st_dev;   /* device */
        nlink_t st_nlink; /* number of links */
        uid_t   st_uid;   /* owner UID */
        gid_t   st_gid;   /* owner GID */
        off_t   st_size;  /* number of bytes*/
        time_t  st_atime; /* last acccess */
        time_t  st_mtime; /* last modification */
        time_t  st_ctime; /* last data modification */
    };
```

STAT – Information on a file

- Check file type in `st_mode`:

`S_ISDIR(s.st_mode)`	Is directory?
`S_ISCHR(s.st_mode)`	Is special character file?
`S_ISBLK(s.st_mode)`	Is special block file?
`S_ISREG(s.st_mode)`	Is regular file?
`S_ISFIFO(s.st_mode)`	Is pipe or FIFO?

UTIME – Altering date attributes

- Service:

```
#include <sys/stat.h>
#include <utime.h>

int utime(char *name, struct utimbuf *times);
```

- Arguments:
  - **name** File name.
  - **times** Structure with last access and modification dates.
    - **time_t actime** Access date.
    - **time_t mctime** Modification date.
- Returns:
  - Returns zero or -1 upon error.
- Description:
  - Change dates for last acces and last moficiation with values of structure **struct utimbuf**

Example: Cpying a file

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>

#define BUFSIZE 512

main(int argc, char **argv) {
  int fd_ent, fd_sal;
  char buffer[BUFSIZE];
  int n_read;
```

```
/* open input file */
  fd_ent = open(argv[1],
   O_RDONLY);
  if (fd_ent < 0)  {
    perror("open");
    exit(-1);
  }

  /* create output file */
  fd_sal = creat(argv[2], 0644);
  if (fd_sal < 0)  {
    close(fd_ent);
    perror("open");
    exit(-1);
  }
```

Example: Copying a file

```
  /* reading loop */
  while ((n_read = read(fd_ent, buffer, BUFSIZE)) > 0)  {
    /* write buffer on output file*/
    if (write(fd_sal, buffer, n_read) < n_read)  {
      perror("write2);
      close(fd_ent); close(fd_sal);
      exit(-1);
    }
  }

  if (n_read < 0)  {
    perror("read");
    close(fd_ent); close(fd_sal);
    exit(-1);
  }
  close(fd_ent); close(fd_sal);
  exit(0);
}
```

POSIX services for directories

- **Logical view:**
  - A directory is a file with records of structure DIR.
  - It can be operated as regular file for reading.
    - Do not write to it with regular writing calls.
- DIR structure:
  - **d_ino;** *// i-node*
  - **d_off;** *// Position in file of element in directory*
  - **d_reclen;** *// Directory size.*
  - **d_type;** *// Element type*
  - **d_name[0];** *// File name **of variable length***

  - **CARE NOTE**: Variable length records cannot be manipulated as fixed size records.
  - **Solution**: System calls to improve directories.

POSIX services for directories

- **DIR *opendir(const char *_dirname_);**
  - Open a directory and return a pointer of type **DIR** to the beginning.
- **int readdir_r(DIR *_dirp_, struct dirent *_entry_, struct dirent **_result_);**
  - Read next directory entry and returns result in a `struct dirent`.
- **long int telldir(DIR *_dirp_);**
  - Get current position of pointer within directory file.
- **void seekdir(DIR *_dirp_, long int _loc_);**
  - Advance from current position to position specified by loc. Never goes backward.
- **void rewinddir(DIR *_dirp_);**
  - Reset file pointer and move it to the beginning.
- **int closedir(DIR *_dirp_);**
  - Close directory file.

Goal accomplished?

- To understand what an operating system service is.
- To understand mechanisms participating in a system call.
- To know POSIX interface features.
- To know main services offered by POSIX.

OPERATING SYSTEMS: INTRODUCTION AND BASIC CONCEPTS

Operating System Services

Projections in POSIX

```
void *mmap(void *direc, size_t len, int prot,
int flags, int fd, off_t offset);
```

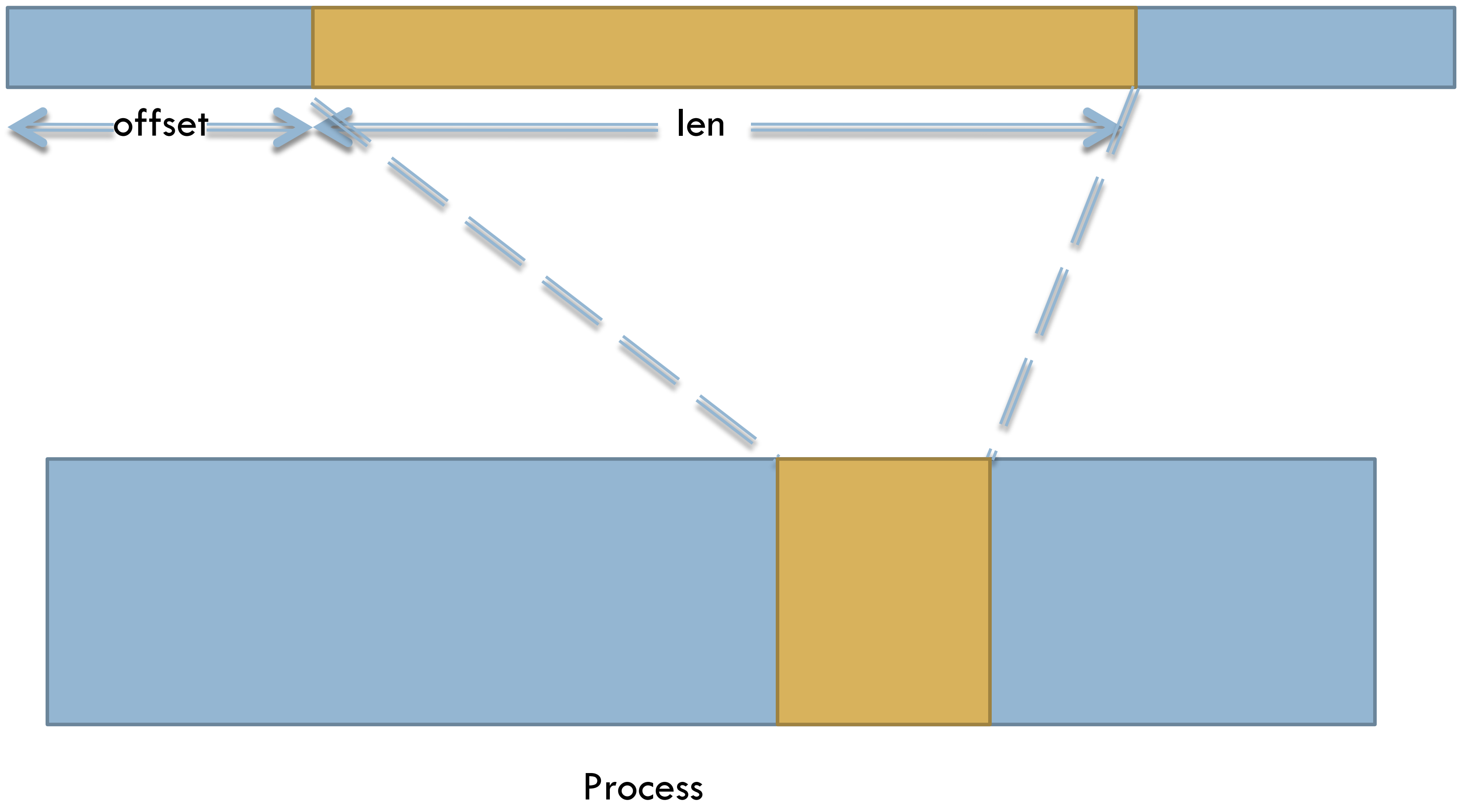
- Sets a projection from a process address space and a file.
  - Return memory address where file was projected.

  - **direc**: address where projection is performed. If NULL OS selects one.
  - **len**: specifies number of bytes to project.
  - **prot**: Protection bits for the area.
  - **flags**: Properties for the region.
  - **fd**: File descriptor to be used in memory.
  - **offset**: Initial offset on the file.

Projections with mmap

- Protection types:
  - **PROT_READ**: Can read.
  - **PROT_WRITE**: Can write.
  - **PROT_EXEC**: Can execute.
  - **PROT_NONE**: Cannot access.
- Properties of a memory region:
  - **MAP_SHARED:**
    - Shared region.
    - Modifications affect to file.
    - Child processes share region.
  - **MAP_PRIVATE:**
    - Private region.
    - File is not modified.
    - Child processes get non-shared duplicates.
  - **MAP_FIXED:**
    - File must be projected in an address specified by the call.

POSIX Projection



POSIX removing mapping

- `void munmap(void *direc, size_t len);`
  - Removes part of the process address space from address **direc** to address **direc + len**

Example: Count number of blanks in file

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main() {
  int fd;
  struct stat dstat;
  int i, n;
  char c,
  char * vec;

  fd = open("datos.txt",O_RDONLY);
  fstat(fd, &dstat);
  vec = mmap(NULL, dstat.st_size,
    PROT_READ, MAP_SHARED, fd, 0);
  close(fd);
  c =vec;
  for (i=0;i<dstat.st_size;i++) {
    if (*c==' ') {
      n++;
    }
    c++;
  }
  munmap(vec, dstat.st_size);
  printf("n=%d,\n",n);
  return 0;
}
```

Example: Copy a file

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main() {
  int i, fd1, fd2;
  struct stat dstat;
  char * vec1, *vec2, *p, *q;

  fd1 = open("f1", O_RDONLY);
  fd2 = open("f2",
   O_CREAT|O_TRUNC|O_RDWR,0640);
  fstat(fd1,&dstat);
  ftruncate(fd2, dstat.st_size);
  vec1= mmap (0, bstat.st_size,
    PROT_READ, MAP_SHARED, fd1,0);
  vec2= mmap (0, bstat.st_size,
    PROT_READ, MAP_SHARED, fd2,0);

  close(fd1); close(fd2);

  p=vec1; q=vec2;
  for (i=0;i<dstat.st_size;i++) {
    *q++ = *p++;
  }

  munmap (fd1, bstat.st_size);
  munmap (fd2, bstat.st_size);

  return 0;
}
```