CSCE 313 — Midterm 2 Cheat Sheet (Comprehensive)

1 Process API & `exec` Pattern

- **Sequence:** `fork()` → `exec*()` → `wait()/waitpid()` → `exit()`
- `execvp()` replaces the process image — it **never returns** if successful.
- Use `waitpid(pid, &status, 0)` to block until child finishes.

Example:

```
pid_t pid = fork();
if (pid == 0) {
  char *args[] = {"ls", "-l", NULL};
  execvp(args[0], args);
  perror("execvp");
  _exit(127);
}
waitpid(pid, NULL, 0);
```

2 Unix File Descriptors (FDs)

- **Standard FDs:** `0 = stdin`, `1 = stdout`, `2 = stderr`
- `open()` returns the **lowest unused fd**.
- File offset and flags live in the **open-file table**.
- `dup2(oldfd, newfd)` duplicates FDs → used for redirection.
- `close(fd)` releases file descriptors to prevent leaks.

Redirect stdout → file

```
int fd = open("out.txt", O_CREAT|O_WRONLY|O_TRUNC, 0644);
dup2(fd, 1);
close(fd);
execvp(argv[0], argv);
```

3 Common `open()` Flags

Flag	Meaning	Typical Use
`O_RDONLY`	Open for read only	reading existing files
`O_WRONLY`	Open for write only	write logs, append output
`O_RDWR`	Read and write	random access files

Flag	Meaning	Typical Use
`O_CREAT`	Create file if it doesn't exist	requires `mode` argument
`O_TRUNC`	Truncate file to 0 length	overwrite existing files
`O_APPEND`	Append writes to end	log files
`O_EXCL`	Fail if file exists (with `O_CREAT`)	atomic create
`O_NONBLOCK`	Non-blocking mode	pipes/sockets
`O_SYNC`	Wait for writes to reach disk	reliability-critical writes

Example: `open("log.txt", O_CREAT|O_WRONLY|O_APPEND, 0644);`

4 Descriptor, File, and Inode Tables

Table	Scope	Contains
FD table	Per-process	integers (0,1,2…) → open-file entries
Open-file table	Kernel-wide	file offset, flags, refcount
Inode/Vnode	System-wide	metadata (mode, size, owner), block pointers

Multiple processes can share an open-file entry (e.g., after `fork()`), but each has its own FD table.

5 Pipes & Pipelines

- `pipe(fd)` → creates unidirectional data channel.
- `p[0]` = read end, `p[1]` = write end.
- Close unused ends in both parent & child.
- Used for connecting processes via stdin/stdout.

Example (`ls | wc -l`):

```
int p[2]; pipe(p);
if (fork()==0){ dup2(p[1],1); close(p[0]); execlp("ls","ls",NULL); }
if (fork()==0){ dup2(p[0],0); close(p[1]); execlp("wc","wc","-l",NULL); }
close(p[0]); close(p[1]);
wait(NULL); wait(NULL);
```

6 Background Jobs (&) and Process Management

- `&` launches child process **without waiting**.
- Parent returns to shell prompt immediately.
- Must still call `waitpid(-1, &st, WNOHANG)` to reap finished children.

Key functions

Function	Description
`fork()`	Create new process
`execvp()`	Replace image with another program
`waitpid(pid,&st,WNOHANG)`	Reap without blocking
`kill(pid,SIGTERM)`	Send signal to a process

Zombie: child exits, parent didn't call `wait()` → "defunct" state.
Orphan: parent exits, child adopted by `init`/`systemd`.

Good practice: install a `SIGCHLD` handler that reaps all children.

7 Threads (Pthreads)

- Threads share address space, heap, globals, and open files.
- Each thread has its own stack and registers.
- Protect shared data with mutexes.

Common functions

Function	Purpose
`pthread_create()`	Start thread
`pthread_join()`	Wait for thread
`pthread_exit()`	End thread explicitly
`pthread_detach()`	Run thread without joining
`pthread_mutex_lock()` / `unlock()`	Protect shared data

Example:

```
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
int counter = 0;
void *add(void *arg){
  for(int i=0;i<100000;i++){
    pthread_mutex_lock(&m);
    counter++;
    pthread_mutex_unlock(&m);
  }
  return NULL;
}
```

8 Condition Variables & Synchronization

- Allow threads to sleep until a condition becomes true.
- Always pair with a mutex.
- Use a `while` loop to avoid spurious wakeups.

Wait pattern

```
pthread_mutex_lock(&m);
while(!ready)
  pthread_cond_wait(&cv,&m);
pthread_mutex_unlock(&m);
```

Producer–Consumer pattern

```
pthread_mutex_lock(&m);
while(buffer_full()) pthread_cond_wait(&not_full,&m);
enqueue(item);
pthread_cond_signal(&not_empty);
pthread_mutex_unlock(&m);
```

Deadlock prevention

1. Lock in a consistent order.
2. Hold locks only briefly.
3. Avoid circular dependencies.

9 File Permissions, Ownership & Special Bits

Bit	Applies To	Meaning
`r`	File/Dir	Read bytes / list directory
`w`	File/Dir	Write bytes / create-delete entries
`x`	File/Dir	Execute / traverse directory

Sticky Bit (`+t`): only file owner or root can delete inside directory (e.g., `/tmp`).
Example: `drwxrwxrwt` → shared temp directory.

setuid & setgid (Privilege Bits)

- **Purpose:** let users run programs with the owner or group's privileges.
- **setuid:** process runs with **EUID = file owner**.
- **setgid:** process runs with **EGID = file group**.
- **setgid on directories:** new files inherit directory's group.

Notation

Mode	Meaning
`-rwsr-xr-x`	setuid active (user `s`)
`-rwxr-sr-x`	setgid active (group `s`)
`drwxrwsr-x`	directory with setgid (shared group)

Examples

- `-rwsr-xr-x root root /usr/bin/passwd` → runs as root.
- `drwxrwsr-x root staff /project/shared` → files group = staff.

Security

- Works only on binaries.
- Avoid setuid on user-owned scripts.
- Common safe uses: `passwd`, `sudo`, `ping`.

---

10 Filesystem Inodes & Block Pointers

- **Inode** = metadata + pointers to data blocks.
- Pointers are used to access file data stored on disk.

Pointer Type	Count	Capacity (4 KB blocks)
Direct	12	12 × 4 KB = 48 KB
Single Indirect	1	1024 × 4 KB = 4 MB
Double Indirect	1	1024^2 × 4 KB ≈ 4 GB
Triple Indirect	1	1024^3 × 4 KB ≈ 4 TB

How to calculate inode capacity

1. **Direct:** `#direct × block_size`
   → `12 × 4 KB = 48 KB`
2. **Single indirect:** `block_size / ptr_size × block_size`
   → `(4096 / 4) × 4096 = 4 MB`
3. **Double indirect:** `(block_size / ptr_size)² × block_size`
   → `(1024²) × 4 KB = 4 GB`
4. **Triple indirect:** `(block_size / ptr_size)³ × block_size`
   → `(1024³) × 4 KB = 4 TB`

Path lookup

1. Parse path (`/dir1/dir2/file`) → traverse directories.
2. Find target inode → read block pointers.
3. Access data through direct/indirect blocks.

Links

- **Hard link:** another directory entry pointing to same inode.
- **Soft link (symlink):** separate inode storing path text.

---

11 Signals Table

#	Name	Default	Typical Cause / Meaning
1	SIGHUP	T	Terminal hangup / reload config
2	SIGINT	T	Ctrl-C interrupt
3	SIGQUIT	C	Ctrl-\ (quit, core dump)
4	SIGILL	C	Illegal instruction
5	SIGTRAP	C	Debug breakpoint
6	SIGABRT	C	Abort (abort())
7	SIGBUS	C	Bus error
8	SIGFPE	C	Divide by zero / math fault
9	SIGKILL	T	**Uncatchable kill**
10	SIGUSR1	T	User-defined
11	SIGSEGV	C	Segmentation fault
12	SIGUSR2	T	User-defined
13	SIGPIPE	T	Write to pipe w/ no reader
14	SIGALRM	T	Timer expired
15	SIGTERM	T	Graceful termination
17	SIGCHLD	I	Child terminated
18	SIGCONT	Cnt	Continue if stopped
19	SIGSTOP	S	**Uncatchable stop**
20	SIGTSTP	S	Ctrl-Z
28	SIGWINCH	I	Window resize
29	SIGIO	T	I/O possible
30	SIGPWR	T	Power failure
31	SIGSYS	C	Bad syscall

☑ Exam-Day Checklist

- ☐ Wire **multi-stage pipelines** with `dup2` and close order.
- ☐ Know which process does `> outfile` (**last stage**).

- ☐ Explain **FD table vs file table vs inode**.
- ☐ Implement a `SIGCHLD` **reaper** and explain zombies.
- ☐ Recall directory `r/w/x` semantics + **sticky bit**.
- ☐ Estimate **direct (≈48KB)** and **indirect (≈4MB)** capacity @ 4KB blocks.
- ☐ Condition variable loop: `while (!cond) pthread_cond_wait(...)`.