# Assignment-2

Shivansh Garg (2024MCS2450) Rohit Patidar (2024JCS2042)

April 2025

## Individual Control Flow Explanation

This section explains the control flow for each signal triggered by the key combinations Ctrl+C, Ctrl+B, Ctrl+F, and Ctrl+G.

### 1. Ctrl+C → SIGINT (Terminate)

1. **User Input:** User presses Ctrl+C.
2. **Shell Input Handling:**
   - The shell detects Ctrl+C and interprets it as a command to terminate processes.
   - It calls `send_signal_to_all(SIGINT);` to broadcast the termination signal.
3. **Inside send_signal_to_all():**
   - Iterates through all processes.
   - For each process (except shell and init):
     - Sets `p->killed = 1`
     - If state is `SLEEPING`, `SUSPENDED`, etc., sets `p->state = RUNNABLE` so that the process wakes up and gets killed in the next scheduler cycle.
4. **Result:** Processes are terminated.

### 2. Ctrl+B → SIGBG (Send to Background)

1. **User Input:** User presses Ctrl+B.
2. **Shell Input Handling:**
   - Shell calls `send_signal_to_all(SIGBG);` to suspend all other processes.
3. **Inside send_signal_to_all():**
   - Iterates over processes (except shell and init).
   - For eligible processes in RUNNING, RUNNABLE, etc.:
     - Sets `p->state = SUSPENDED`
     - Sets `p->suspended = 1`
     - Reassigns their parent to `initproc` (if pid > 2)
4. **Result:** All active processes are suspended and sent to background.

**3. Ctrl+F → SIGFG (Bring to Foreground)**

1. **User Input:** User presses Ctrl+F.
2. **Shell Input Handling:**
   - Shell calls `send_signal_to_all(SIGFG);` to resume suspended processes.
3. **Inside send_signal_to_all():**
   - Iterates over all processes.
   - For each process where `p->state == SUSPENDED`:
     - Sets `p->state = RUNNABLE`
     - Sets `p->suspended = 0`
4. **Result:** Suspended processes resume execution.

**4. Ctrl+G → SIGCUSTOM (Custom Signal)**

1. **User Input:** User presses Ctrl+G.
2. **Shell Input Handling:**
   - Shell calls `send_signal_to_all(SIGCUSTOM);` to notify all processes.
3. **Inside send_signal_to_all():**
   - Iterates over processes.
   - If a process has a custom handler (`p->signal_handler != 0`):
     - Sets `p->pending_signal = SIGCUSTOM`
     - If `p->state == SLEEPING`, wakes it up by setting `RUNNABLE`
4. **Result:** When scheduled, process checks for `pending_signal` and invokes its handler.

## Dynamic Priority Scheduling: Effect of $\alpha$ and $\beta$ Parameters

The scheduling algorithm implemented uses a dynamic priority function to determine which process should be scheduled next. The priority $\pi_i(t)$ of process $P_i$ at time $t$ is defined as:

$$\pi_i(t) = \pi_i(0) - \alpha \cdot C_i(t) + \beta \cdot W_i(t)$$

- $\pi_i(0)$: Initial priority of process $P_i$ (defined in `Makefile`)
- $C_i(t)$: Total CPU time consumed by $P_i$ up to time $t$
- $W_i(t)$: Waiting time of $P_i$ since its creation
- $\alpha$, $\beta$: Tunable parameters to control priority decay and boost (defined in `Makefile`)

### Interpretation of Parameters

- **Parameter** $\alpha$: Controls how quickly a process loses priority based on its CPU usage.
  - A higher $\alpha$ causes CPU-bound processes to lose priority faster.
  - Helps prevent long-running CPU-intensive tasks from hogging the CPU.
- **Parameter** $\beta$: Controls how much waiting time contributes to the priority.
  - A higher $\beta$ rewards processes that have waited longer.
  - Helps I/O-bound and short processes gain CPU access sooner.

### Effect on Different Process Types

### Experimental Observation

In the profiling experiments conducted:

- With **higher** $\alpha$ **(e.g., 3)** and **moderate** $\beta$ **(e.g., 2)**:
  - CPU-bound jobs were penalized after initial execution.
  - Short/I/O-bound jobs were scheduled more frequently and completed quickly.
  - Fairness was observed with no starvation.
- With **lower** $\alpha$ **(e.g., 1)** and **higher** $\beta$ **(e.g., 4)**:
  - Even minor waiting resulted in a large priority boost.
  - I/O-bound processes preempted CPU-bound ones very frequently.
  - Potential starvation of long CPU-intensive jobs was observed.

### Conclusion

The $\alpha$ and $\beta$ parameters significantly impact the system's responsiveness and fairness:

- To favor **interactive/I/O-bound jobs** and prevent starvation, use a **higher** $\beta$.
- To prevent **long-running CPU-bound tasks from dominating**, use a **higher** $\alpha$.
- An optimal balance ensures all processes get CPU access proportionately based on their needs.

| **Process Type** | **High $\alpha$, Low $\beta$** | **Low $\alpha$, High $\beta$** |
|---|---|---|
| CPU-bound | Quickly penalized due to high CPU usage. Low chances of frequent scheduling. | Deprioritized slowly. May hog the CPU. |
| I/O-bound | May starve if $\beta$ is too low. | Frequently scheduled due to accumulated wait time. |
| Short jobs | May not get enough priority once CPU usage increases. | Scheduled quickly, finish early. |

Table 1: Effect of $\alpha$ and $\beta$ on process behavior