Report : Assignment 2

Disha (2022CS11118)

Signal Flow and Key Handling

```
case C('C'):  // Ctrl+C (SIGINT)
  consputs("Ctrl-C is detected by xv6\n");
  keyinterrupt(SIGINT);
  // ctrl_c = 1;
  break;
case C('B'):
consputs("Ctrl-B is detected by xv6\n");
  keyinterrupt(SIGBG);
  break;

case C('F'):
  consputs("Ctrl-F is detected by xv6\n");
  keyinterrupt(SIGFG);
  break;

case C('G'):
  consputs("Ctrl-G is detected by xv6\n");
  keyinterrupt(SIGCUSTOM);
  break;
```

1. **Ctrl + C (SIGINT)**:
   - When the user presses **Ctrl + C**, **consoleintr()** detects it.
   - The **SIGINT** signal is passed to **keyinterrupt()** in **proc.c**.
   - Inside **keyinterrupt()**, the kernel marks all processes (with **pid > 2**) as **killed**.
   - The **trap.c** function checks if the `killed` flag is set. If it is, the `exit()` function is called, which terminates the process and cleans up its memory.
   - The parent process of the terminated process receives **-1** from the `wait()` system call, indicating that the child process has been killed.

2. **Ctrl + B (SIGBG)**:
   - When **Ctrl + B** is pressed, **consoleintr()** sends the **SIGBG** signal to **keyinterrupt()**.
   - In **keyinterrupt()**, the kernel marks processes with **pid > 2** as **suspended**. The suspended processes are not scheduled.
   - These suspended processes return **-2** from `wait()`, which tells the parent not to wait for them.
   - The **scheduler** checks for the `suspended` flag and skips these processes until they are resumed.
   - Now the parent, ie the shell (pid 2) is woken up.

```
}

if(myproc() && myproc()->suspended)
  wakeup(myproc()->parent);
```

3. **Ctrl + F (SIGFG)**:
   - Pressing **Ctrl + F** sends the **SIGFG** signal to the kernel via **consoleintr()** and **keyinterrupt()**.
   - Inside **keyinterrupt()**, the suspended flag for the relevant processes is cleared, allowing them to continue running.
   - The **trap.c** function is used to bring the suspended process back into the foreground.
4. **Ctrl + G (SIGCUSTOM)**:
   - Pressing **Ctrl + G** triggers the **SIGCUSTOM** signal, which is detected by **consoleintr()**.

```
int
sys_signal(void)
{
  sighandler_t handler;
  if(argptr(0, (void*)&handler, sizeof(handler)) < 0)
    return -1;
  myproc()->sighandler = handler;
  return 0;
}
```

   - The **pending_custom_signal** is set to the SIGCUSTOM inside only if the handler is registered.
   - In **trap.c**, if the **pending_custom_signal** is set, it is invoked to handle the custom signal.

```
case SIGCUSTOM:
  if (p->sighandler) {
    p->pending_custom_signal = SIGCUSTOM;
  }
  break;
}
```

```
if(myproc() && (myproc()->pending_custom_signal == SIGCUSTOM)){
  myproc()->tf->esp -= 4;
  *(uint*)(myproc()->tf->esp) = myproc()->tf->eip;
  myproc()->tf->eip = (uint)myproc()->sighandler;
  myproc()->pending_custom_signal = 0;
}
```

Xv6 Scheduler

Custom fork Creation (`custom_fork`)

The **custom_fork** function creates processes with a specified execution time:

```
//custom fork
if((p->pid > 2) && (p->exec_time != -1) && (p->cpu_ticks >= p->exec_time)) {
  p->killed = 1;
}
```

- If **exec_time** is **-1**, the process runs to completion.

- If **exec_time** is a positive value, the process is forcibly terminated in the scheduler once it exceeds the allowed time.

```
  if(p->state == RUNNABLE) {
    p->dynamic_priority = p->init_priority - (ALPHA * p->cpu_ticks) + (BETA * p->wait_ticks);
  }
}
```

Effect of α (ALPHA) on Scheduling

- **ALPHA** helps control how long a process stays on the CPU. If **ALPHA** is higher:
  - Processes that have already used more CPU time are **less likely** to be rescheduled.
  - The scheduler gives priority to processes that have been waiting for CPU time, allowing fairer scheduling between CPU-bound and I/O-bound processes.
- **Scheduling Behavior**: The higher the **ALPHA** value, the less likely CPU-heavy processes are to be selected for execution again. This ensures that waiting processes are given a chance to run.

Effect of β (BETA) on Scheduling

- **BETA** prioritizes processes that have been **waiting** for a long time.
- **Higher BETA** values make long-waiting processes more likely to be scheduled next.
- **Scheduling Behavior**: This helps prevent starvation of processes that have been waiting for a long time in the queue. It gives them a higher chance of running, ensuring fairness in resource allocation.

Test results:

1. When ALPHA was set large (10), and BETA was small(1), the processes which were scheduled once, only got a chance in cyclic order, ie, the first one scheduled was rescheduled after all the processes were scheduled once.
2. When ALPHA was set to a negative large value, then the process that was running initially was the one which got rescheduled.
3. When BETA was set negative and ALPHA was also negative then the process which waited a lot was frequently scheduled.

Overall Scheduler Behavior

CPU-Bound Jobs

When **ALPHA** is higher, the scheduler deprioritizes processes that have already consumed more CPU time.

- **Higher ALPHA = Less Likely to be Rescheduled**: This means that **CPU-bound jobs**, which tend to consume a lot of CPU time, are less likely to be selected for execution when **ALPHA** is set to a higher value.

BETA prioritizes processes that have been waiting for CPU time, i.e., processes with higher waiting times are more likely to be scheduled next.

- **Higher BETA = Less Likely to Favor CPU-Bound Jobs**: Since **BETA** increases the chances of long-waiting processes (like I/O-bound jobs) getting scheduled, **CPU-bound jobs** are less likely to be selected if they have not been waiting for a long time.

I/O-Bound Jobs:

- **ALPHA** impacts how much CPU time a process has already consumed. If **ALPHA** is set to a higher value, the scheduler deprioritizes processes that have used more CPU time.
    - **Lower ALPHA = More Likely to be Rescheduled**: **I/O-bound jobs** typically consume less CPU time since they spend most of their time waiting for I/O. If **ALPHA** is lower, these processes are more likely to be scheduled because they haven't consumed much CPU time.
- **BETA** prioritizes processes that have waited the longest for CPU time.

- **Higher BETA = More Likely to be Scheduled**: Since **I/O-bound jobs** typically spend a lot of time waiting for I/O, they are likely to have high waiting times in the scheduler queue. **Higher BETA** ensures that these processes will have a better chance of getting scheduled, as their waiting time will be taken into account. This allows **I/O-bound jobs** to execute when the CPU is available, rather than being overlooked in favor of **CPU-bound jobs**.