Sistem Operasi (INF-201)

Pertemuan 3: Manajemen Proses & POSIX Process API

Tim Dosen Sistem Operasi

Program Studi Informatika – Universitas Bengkulu

Semester Ganjil 2026/2027

Capaian Pembelajaran (Sub-CPMK 3)

- Mahasiswa mampu menjelaskan konsep Proses, State Proses, dan Process Control Block (PCB).
- Mahasiswa memahami struktur PCB pada kernel nyata (**MIT xv6 Kernel**).
- Mahasiswa memahami mekanisme Context Switch dan Inter-Process Communication (IPC).
- Mahasiswa mampu merancang aplikasi Custom Shell sederhana menggunakan POSIX API (`fork`, `exec`, `wait`).

Konsep Proses & Process Control Block (PCB)

Definisi Proses (OSTEP Ch. 4)

Sebuah program yang sedang dalam kondisi dieksekusi di RAM (*program in execution*), memiliki ruang alamat tersendiri (Stack, Heap, Data, Text).

5 State Proses Utama:

1. **New**: Proses baru dibuat.
2. **Ready**: Siap di-CPU-kan dalam *Ready Queue*.
3. **Running**: Instruksi dieksekusi CPU.
4. **Waiting**: Menunggu I/O / event.
5. **Terminated**: Selesai dieksekusi.

Isi PCB (Kernel Data Structure):

- Process ID (PID) & Parent PID.
- Process State & Program Counter.
- CPU Registers & Kernel Stack Pointer.
- CPU Scheduling Info.
- Memory Table & File Descriptors.

Struktur PCB pada Kernel Nyata (`struct proc` MIT xv6)

Definisi `struct proc` dari MIT xv6 Kernel (`kernel/proc.h`):

```
enum procstate { UNUSED, USED, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };

struct proc {
  struct spinlock lock;        // Lock pelindung struktur proc
  enum procstate state;        // State proses saat ini
  int pid;                     // Process ID
  struct proc *parent;         // Pointer ke proses induk
  uint64 kstack;               // Alamat memori Kernel Stack
  uint64 sz;                   // Ukuran memori proses (bytes)
  pagetable_t pagetable;       // User Page Table
  struct trapframe *trapframe; // Frame untuk menyimpan register saat Trap
  struct context context;      // Register untuk context switch
  struct file *ofile[NOFILE];  // Tabel berkas yang sedang dibuka
};
```

POSIX System Calls: Pembuat & Pengelola Proses

- `fork()`: Menduplikasi proses pemanggil (anak memiliki ruang memori terpisah).
- `execvp()`: Mengganti memori proses saat ini dengan program executable baru.
- `wait() / waitpid()`: Induk menunggu proses anak selesai (mencegah *Zombie*).

Contoh Kode C: Process Creation & Execution

```
pid_t pid = fork();
if (pid == 0) {
    // Proses Anak: Ganti program dengan /bin/ls
    char *args[] = {"ls", "-l", NULL};
    execvp(args[0], args);
} else if (pid > 0) {
    // Proses Induk: Tunggu anak selesai
    wait(NULL);
    printf("Proses␣anak␣telah␣selesai␣dieksekusi.\n");
}
```

Tugas Terstruktur 1: Custom Linux Shell

Praktikum 1 (Tugas Terstruktur - Bobot 5%)

Rancanglah sebuah program C bernama `myshell.c` yang bertindak sebagai terminal shell interaktif dengan ketentuan:

1. Menampilkan prompt `myshell>` .
2. Membaca perintah pengguna (misal: `ls -l`, `ps`, `whoami`).
3. Memanggil `fork()` dan `execvp()` untuk mengeksekusi perintah tersebut.
4. Menangani perintah built-in `exit` dan `cd`.

Tugas Membaca Mingguan (Reading Assignment)

Persiapan Pertemuan 4: Penjadwalan CPU (CPU Scheduling)

Sebelum mengikuti kuliah minggu depan, mahasiswa wajib membaca:

- **[OSTEP]** Chapter 7: *Scheduling: Introduction* (hal. 61–72)
- **[OSTEP]** Chapter 8: *Scheduling: The Multi-Level Feedback Queue* (hal. 73–84)
- **[OSTEP]** Chapter 10: *Multiprocessor Scheduling* (hal. 97–108)

Pertanyaan Pemicu Diskusi Mandiri

Mengapa algoritma Round Robin sangat baik untuk *Response Time* tetapi buruk untuk rata-rata *Turnaround Time*?