Sistemas Operativos Avanzados

CC-571

César Lara Avila

Universidad Nacional de Ingeniería

(actualización: 2020-06-05)

Bienvenidos

Programa Análitico

1. Virtualización.
2. Tópicos de concurrencia y persistencia.
3. Sistemas distribuidos.
4. Otros Sistemas Operativos

Bibliografía

1. Thomas Anderson, Michael Dahlin, **Operating Systems: Principles and Practice (Volume 1 of 4)**. Recursive Books; 2 edition 2015.

2. Martin Kleppmann, Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'reilly 2017.

3. Remzi Arpaci-Dusseau, Andrea Arpaci-Dusseau, **Operating Systems: Three Easy Pieces**. Arpaci-Dusseau Books 2018.

4. Wyatt S. Newman, **A systematic Approach to Learning Robot with ROS**. CRC Press Taylor & Francis Group 2018.

5. Ali Sunyaev, Internet Computing: Principles of Distributed Systems and Emerging Internet-Based Technologies. Springer; 1st ed. 2020 edition.

Evaluaciones

- Prácticas calificadas. [1]
- Examen parcial y final.
- Cuestionarios y lecturas semanales.
- Tareas de codificación. [2]
- Desarrollo de proyectos grupales. [3]

[1] El número de prácticas calificadas es 4 y son parte de las tareas que el curso tiene a presentar.

[2] Revisa el funcionamiento de los **kernels para Jupyter Notebook** en particular: **jupyter-c-kernel**, **xeus-cling** y los entornos de python, **colab** o **binder** Los lenguajes que se usarán en el curso son C, C++ y Python principalmente.

[3] El desarrollo de trabajos grupales forma parte del Proceso de Acreditación de la Especialidad.

Metodología y dinámica

- El curso es de naturaleza teórico y práctico. Los participantes deben completar los fragmentos de código entregados, como parte de su participación en las sesiones que se les solicite.
- Se les solicitará la activación de su cámara web para los exámenes parcial y final.
- Cualquier consulta o duda será por el tablón del curso o por correo electrónico a la mayor brevedad posible.

Sesión 1

Virtualización

Temario de la sesión 1

- Virtualización.
- La abstracción: Procesos.
- API de Proceso.
- Estructura de datos.

Virtualización

- Para un CPU, la virtualización es tomar una única CPU y hace que se vea como muchas CPU virtuales para las aplicaciones que se ejecutan en el sistema.
- ¿ Cómo puede el sistema operativo proporcionar la ilusión de un suministro casi interminable de dichas CPU?.
- Tiempo compartido: ejecutar un proceso, luego detenerlo y ejecutar otro. El costo potencial es el rendimiento.
- Mecanismos, políticas.

La abstracción: Procesos

- Procesos.

```
ps ax | wc
top
cat /proc/cpuinfo | grep 'model name'
```

- Espacio de direcciones
- Registros
  - Puntero de instrucción (IP).
  - Puntero de pila.
  - Puntero del frame.
- Los programas a menudo también acceden a dispositivos de almacenamiento persistente.
- Políticas y mecanismos separados.

API de Proceso

- Estas API están en cualquier sistema operativo moderno.
  - **Create**
    - Crea un nuevo proceso para ejecutar un programa.
  - **Destroy**
    - Detiene un proceso desbocado.
  - **Wait**
    - Espera a que un proceso deje de ejecutarse.
  - **Miscellaneous Control**
    - Algún tipo de método para suspender un proceso y luego reanudarlo.
  - **Status**
    - Obtiene información de estado sobre un proceso.

Creación de Procesos

1. Se carga código del programa en la memoria, en el espacio de direcciones del proceso.
    - Los programas residen inicialmente en el disco en formato ejecutable.
    - El sistema operativo realiza el proceso de carga de forma perezosa.
2. Se asigna la pila de tiempo de ejecución del programa.
    - Use la pila para variables locales, parámetros de función y dirección de retorno.
    - Inicializa la pila con argumentos --> **argc** y **argv** de la función **main().**
3. Se crea el *heap* del programa.
    - Se utiliza para datos asignados dinámicamente solicitados.
    - El programa pide dicho espacio llamando a **malloc()** y lo libera llamando a **free().**

...continuación

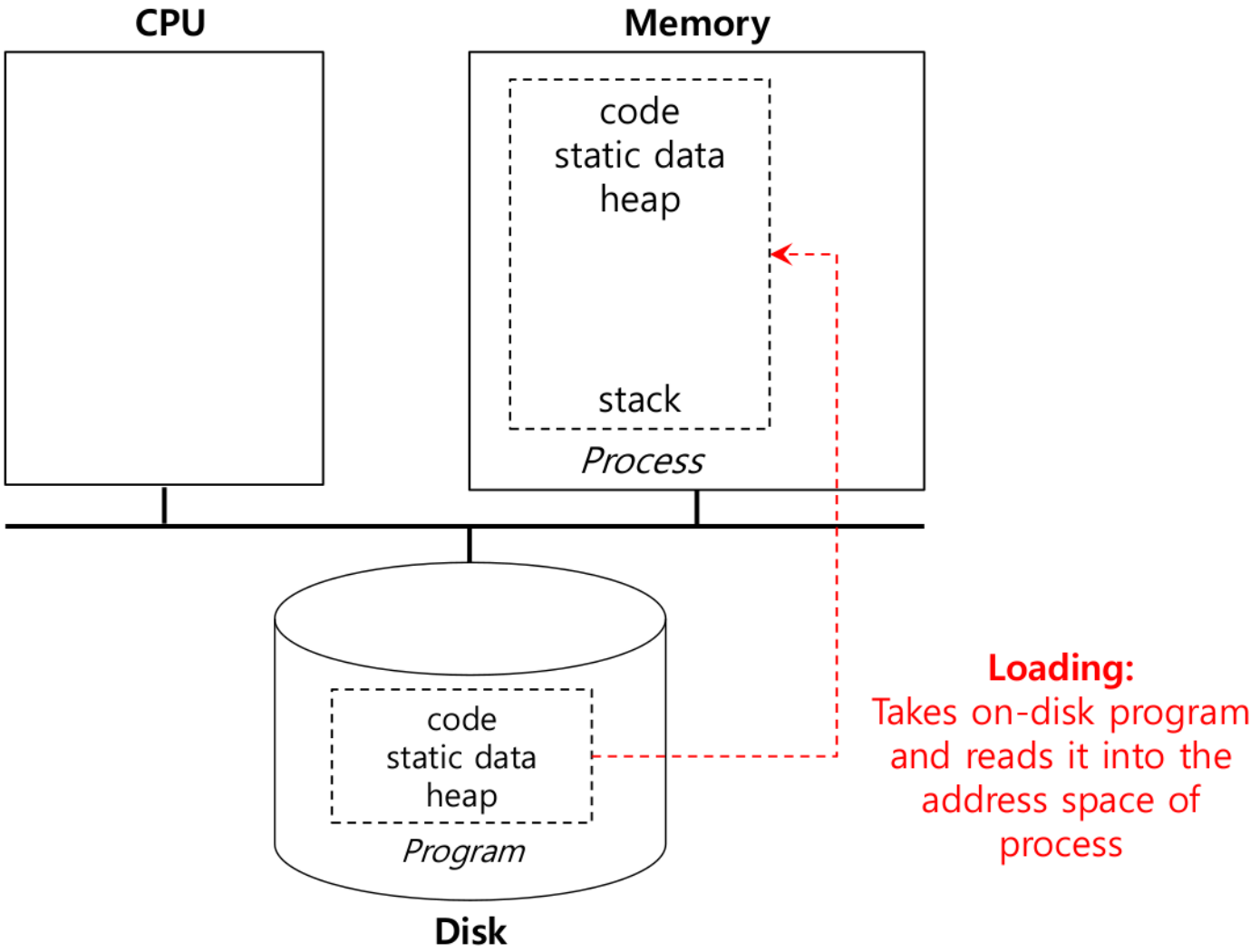
4 . El sistema operativo hace algunas otras tareas de inicialización.

- Configuración de entrada/salida.
  - Cada proceso por defecto tiene tres descriptores de archivo abiertos. Entrada estándar, salida y error

5 . Inicia el programa que se ejecuta en el punto de entrada, es decir, **main()**.

- El sistema operativo transfiere el control de la CPU al proceso recién creado.

Secuencia de carga de un programa a un proceso

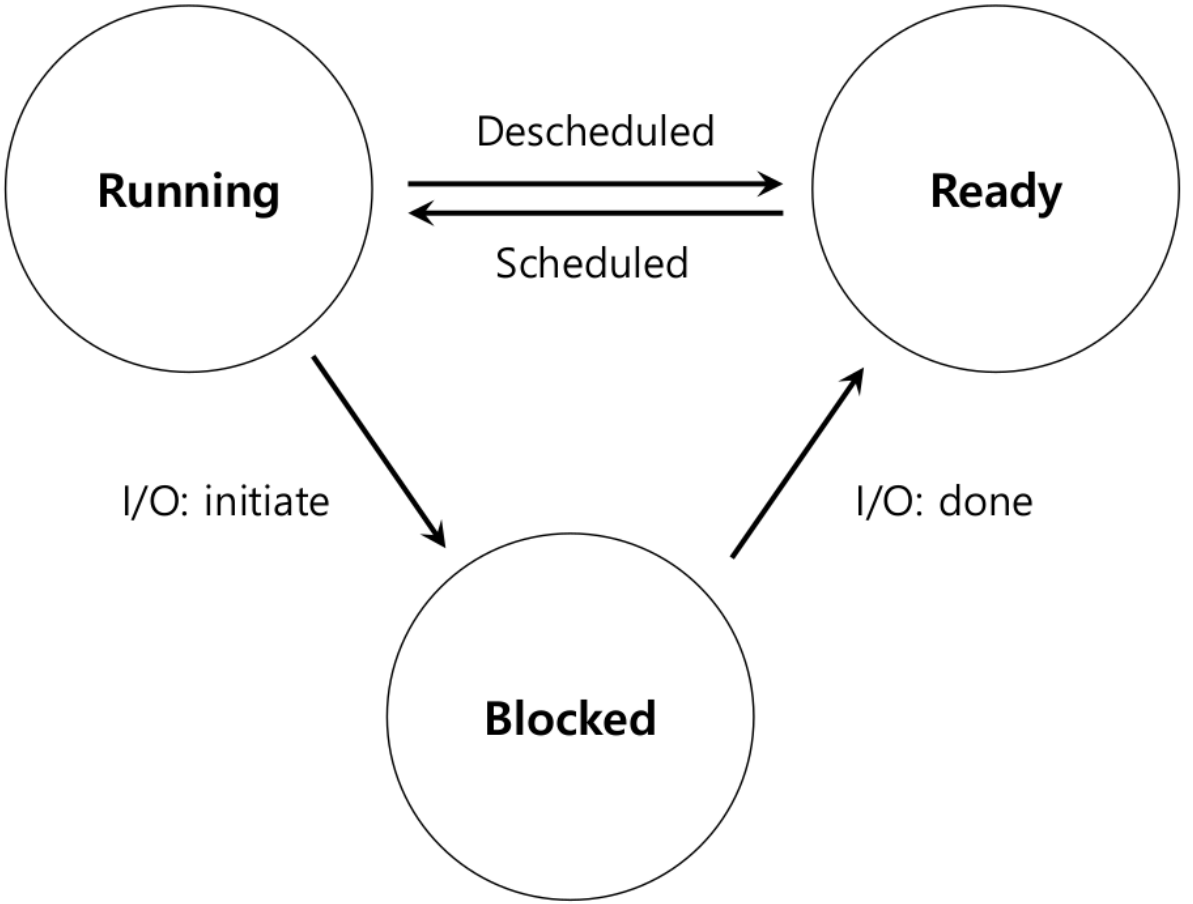


Estados de proceso

Un proceso puede ser uno de tres estados.

- **Running** (en ejecución)
  - Se está ejecutando un proceso en un procesador.
- **Ready** (listo)
  - Un proceso está listo para ejecutarse, pero por alguna razón el sistema operativo ha optado por no ejecutarlo en este momento dado.
- **Blocked** (bloqueado)
  - Un proceso ha realizado algún tipo de operación.
  - Cuando un proceso inicia una solicitud de E/S en un disco, se bloquea y, por lo tanto, algún otro proceso puede usar el procesador.

Proceso: transiciones de estado



Ejercicio

Explica el tracing del estado de procesos.

Time	$Process_0$	$Process_1$	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	
4	Running	Ready	$Process_0$ now done
5	–	Running	
6	–	Running	
7	–	Running	
8	–	Running	$Process_1$ now done

Time	$Process_0$	$Process_1$	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	$Process_0$ initiates I/O
4	Blocked	Running	$Process_0$ is blocked,
5	Blocked	Running	so $Process_1$ runs
6	Blocked	Running	
7	Ready	Running	I/O done
8	Ready	Running	$Process_1$ now done
9	Running	–	
10	Running	–	$Process_0$ now done

Lectura

- **What is the scheduler in an operating system?.**

Estructura de datos

El sistema operativo tiene algunas estructuras de datos clave que rastrean varios datos relevantes.

- Lista de procesos
  - Procesos listos
  - Procesos bloqueados
  - Proceso en ejecución actual
- Contexto de registro

PCB (bloque de control de proceso)

- Una estructura C que contiene información sobre cada proceso.

La estructura de proceso del kernel xv6

```
// the registers xv6 will save and restore
// to stop and subsequently restart a process
struct context {
    int eip;   // Index pointer register
    int esp;   // Stack pointer register
    int ebx;   // Called the base register
    int ecx;   // Called the counter register
    int edx;   // Called the data register
    int esi;   // Source index register
    int edi;   // Destination index register
    int ebp;   // Stack base pointer register
};

// the different states a process can be in
enum proc_state { UNUSED, EMBRYO, SLEEPING,
                  RUNNABLE, RUNNING, ZOMBIE };
```

... continuación

```
// the information xv6 tracks about each process
// including its register context and state
struct proc {
    char *mem;                  // Start of process memory
    uint sz;                    // Size of process memory
    char *kstack;               // Bottom of kernel stack
                                // for this process
    enum proc_state state;      // Process state
    int pid;                    // Process ID
    struct proc *parent;        // Parent process
    void *chan;                 // If non-zero, sleeping on chan
    int killed;                 // If non-zero, have been killed
    struct file *ofile[NOFILE]; // Open files
    struct inode *cwd;          // Current directory
    struct context context;     // Switch here to run process
    struct trapframe *tf;       // Trap frame for the
                                // current interrupt
};
```

Fin!

[1] Todos los gráficos de esta presentación se basan en el texto de Remzi Arpaci-Dusseau y Andrea Arpaci-Dusseau, **Operating Systems: Three Easy Pieces**.