

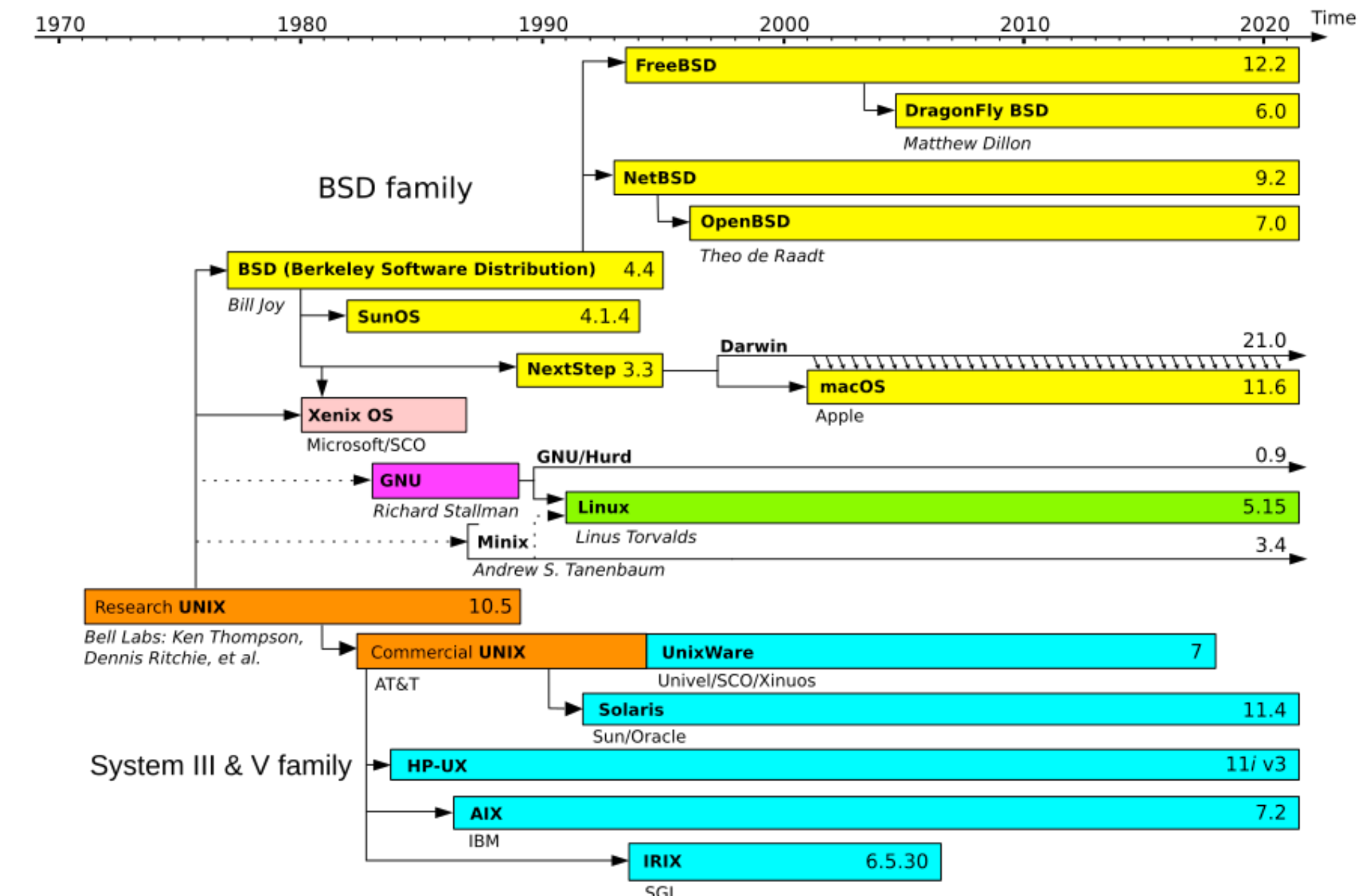
Unix insights: System V vs. POSIX IPC

Laboratorio di Sistemi Operativi - Turno T1

19 novembre 2024

Cenni storici: le “Unix Wars”

- Nel 1969, il sistema operativo **UNIX** nasce nel centro di ricerca dei Bell Labs; negli anni successivi viene distribuito su licenza e trova da subito grande diffusione
- Negli anni '80 si trovano in circolazione 2 grandi filoni di sviluppo:
 - *System V* [AT&T, commerciale]
 - *Berkeley Software Distribution* (BSD) [Berkeley, non commerciale]
- Ciascuno è un ombrello per tutta una serie di sotto-versioni... **Ci sono problemi di incompatibilità!**



POSIX

- **POSIX** (Portable Operating System Interface) è il nome dato a una famiglia di standard, definita a fine anni '80
- Questi standard si occupano di definire un'interfaccia comune, agnostica rispetto al produttore del sistema specifico
- Le definizioni riguardano gestione dei thread, comandi e utilities, direttive per la portabilità e, tra il resto, *inter-process communication* (IPC)


System V vs. POSIX IPC

- Gli strumenti IPC di **System V** sono in uso da più tempo e da un bacino d'utenza più ampio; gli IPC **POSIX**, d'altro canto, sono definiti sulla base dell'esperienza passata e vogliono quindi essere più facili all'uso
- Strumenti di base uguali: *code di messaggi, semafori, memoria condivisa*
 - *POSIX* identificatori testuali (nella forma */nome*), SystemV usa chiavi numeriche
 - *POSIX* è thread-safe, SysV non lo è (**N.B.:** thread != processi)
- Esempio di QoL: le code di messaggi POSIX possono **notificare** un processo dell'arrivo di un messaggio, il quale definisce come comportarsi (*tramite function pointer*) quando arriva un messaggio

Un esempio in POSIX: Lettore e Scrittore

Definizione di semafori e memoria condivisa

```
#define SHM_COMMON "/shm/common"  
#define SEM_WRITER "/sem/writer"  
#define SEM_READER "/sem/reader"  
#define BUF_SIZE 128
```

Two red arrows originate from the code blocks. One arrow points from the `SHM_COMMON` definition to the `shm_open` call in the top-right block. The other arrow points from the `SEM_READER` definition to the `sem_open` call for the reader in the bottom-right block.

```
int fd = shm_open(SHM_COMMON, O_CREAT | O_RDWR, 0666);  
ftruncate(fd, BUF_SIZE);
```

```
sem_t * reader = sem_open(SEM_READER, O_CREAT, 0666, 0);  
sem_t * writer = sem_open(SEM_WRITER, O_CREAT, 0666, 1);
```

Un esempio in POSIX: Lettore e Scrittore

Mapping sullo spazio di indirizzamento del processo

```
char * buf = (char *) mmap(NULL, BUF_SIZE,  
    PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);  
close(fd);
```

- La system call `shm_open()` ritorna un *file descriptor*, che posso usare per mappare la memoria condivisa sullo spazio di indirizzamento locale del processo
- `MAP_SHARED` permette di vedere le modifiche apportate da altri processi

Un esempio in POSIX: Lettore e Scrittore

Utilizzo di semafori e memoria condivisa

Figlio (lettore)

```
if(!(child_pid = fork())) {
    while(!sem_wait(reader)) {
        printf("Child has read access: %s\n", buf);
        bzero(buf, BUF_SIZE);
        sem_post(writer);
    }
}
```

Padre (scrittore)

```
} else {
    int i = 0;
    while(!sem_wait(writer)) {
        sleep(1);
        printf("Parent has write access\n");
        sprintf(buf, "Hello from parent %d\n", i++);
        sem_post(reader);
    }
}
```

- Il segmento di memoria condivisa viene usato come un puntatore normale da entrambi i processi;
- `sem_wait()` e `sem_post()` si comportano come `reserveSem()` e `releaseSem()`
 - Minore flessibilità rispetto a `semop()`
 - Non si può rimanere in *wait-for-zero*

Un esempio in POSIX: Lettore e Scrittore

Chiusura e rimozione delle risorse IPC

```
sem_close(reader);  
sem_close(writer);
```

```
sem_unlink(SEM_READER);  
sem_unlink(SEM_WRITER);  
shm_unlink(SHM_COMMON);
```

- Un processo che non usa più un semaforo lo chiude: `sem_close()`
- Per rimuovere definitivamente un IPC dal sistema: `xxx_unlink()`

Conclusioni

- **In generale, quale usare?** POSIX è più recente, ma supportato da un minor numero di sistemi rispetto a System V
 - Molte codebases esistenti adottano System V, ed eseguire la migrazione agli standard POSIX semplicemente non vale la pena
- **Nel progetto, quale usare?** *System V!* Questo è solo un approfondimento
- **Fonti:**
 - [Un interessante thread su StackOverflow](#)
 - [Una presentazione di linux.conf.au](#)
 - [Wikipedia](#)
 - Esempio del lettore e scrittore, con codice commentato (Moodle)