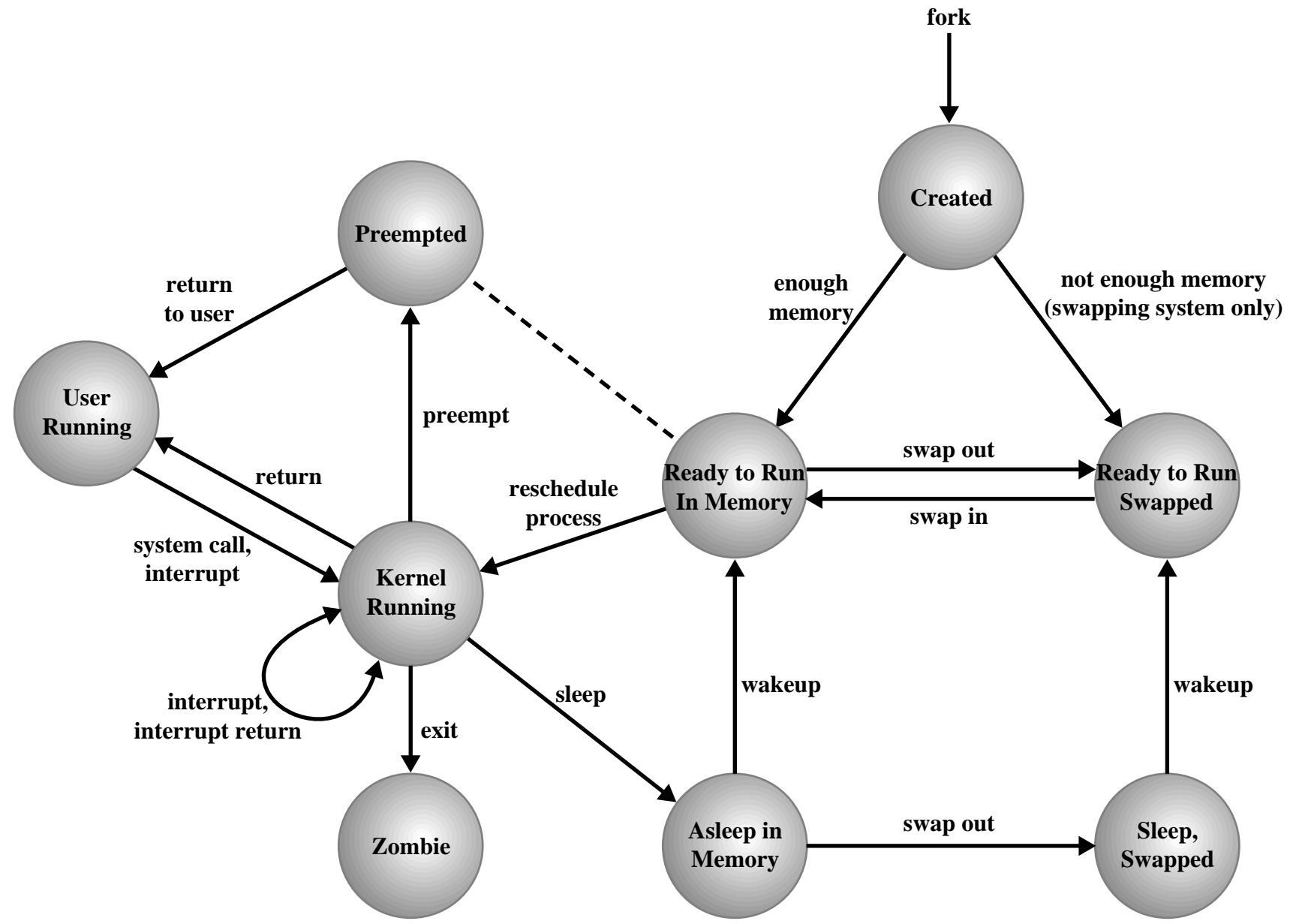


Approfondimento su Gestione dei processi in Unix

- Diagramma di stati di un processo Unix
- Esempio di ciclo di vita
- Algoritmo di gestione delle interruzioni
- User e Kernel mode
- Livelli di contesto, trap ed interrupt

Diagramma degli stati di un processo in UNIX



Ciclo di vita di un processo Unix

- Il processo entra nel sistema nello stato *Created* quando il padre crea il processo tramite la chiamata a `fork`
- Il processo muove in uno stato in cui è pronto a partire, ad es., *Ready to run in memory*
- Se viene selezionato dallo scheduler muove nello stato *Kernel running* (esecuzione in Kernel mode) dove termina la sua parte di `fork`.
- Quando termina l'esecuzione della chiamata di sistema può passare allo stato *User running* (esecuzione in modo User) oppure potrebbe passare nello stato *Preempted* (cioè lo scheduler ha selezionato un'altro processo)
- Dallo stato *User running* può passare allo stato *Kernel running* a seguito di un'altra chiamata di sistema oppure di un'interrupt, ad es., di clock

- Dopo aver eseguito la routine di gestione dell'interrupt di clock il Kernel potrebbe schedulare un altro processo da mandare in esecuzione e quindi il processo in questione passerebbe allo stato *Preempted*
- Lo stato *Preempted* enfatizza il fatto che i processi Unix possono essere prelazionati solo quando tornano da modo Kernel a modo utente
- Se durante l'esecuzione in modo Kernel (ad es. di una system call) il processo deve eseguire operazioni di I/O passa allo stato *Asleep in memory* per essere risvegliato successivamente e passare nello stato *Ready to run*
- Gli stati con etichetta *Swapped* corrispondono a situazioni nelle quali il processo non è più fisicamente in memoria centrale (ad es. non c'è abbastanza memoria per gestire i processi in multitasking)

User and Kernel Mode

- I processi Unix possono operare in modo user e kernel: cioè il kernel esegue nel contesto di un processo le operazioni per gestire chiamate di sistema e interrupt
 - Alla partenza del sistema il codice del kernel viene caricato in memoria principale (con strutture dati (tabelle) necessarie per mappare indirizzi virtuali kernel in indirizzi fisici)
 - Un processo in esecuzione in modo user non può accedere allo spazio di indirizzi del kernel
 - Quando un processo passa ad eseguire in modo kernel tale vincolo viene rilasciato: in questo modo si può eseguire codice del kernel (routine di gestione di interrupt/codice di una chiamata di sistema) nel contesto del processo utente
- Il contesto di un processo: contesto utente (codice, dati, stack), registri, e contesto kernel (entry nella tabella dei processi, u-area, stack kernel)

Livelli di contesto

- La parte dinamica del contesto di un processo (kernel stack, registri salvati) è organizzata a sua volta come stack con un numero di posizioni che dipende dai livelli di interrupt diversi ammessi nel sistema
- Ad esempio se il sistema gestisce interrupt software, interrupt di terminali, di dischi, di tutte le altre periferiche, e di clock: avremo al più sette livelli di contesto
 - Livello 0: User
 - Livello 1: Chiamate di sistema
 - Livelli 2-6: Interrupt (l'ordine dipende dalla priorità associata alle interrupt)

Esempio di esecuzione nel contesto di un processo

- Il processo esegue una chiamata di sistema: il kernel salva il suo contesto (registri, program e stack pointer) nel livello 0 e crea il contesto di livello 1
- La CPU riceve e processa un interrupt di disco (il controllo viene fatto prima dell'esecuzione della prossima istruzione): il kernel salva il contesto di livello 1 (registri, stack kernel) e crea il livello 2 nel quale si esegue la routine di gestione dell'interrupt di disco
- La CPU riceve un interrupt di clock: il kernel salva il contesto di livello 2 (registri, stack kernel per la routine di gestione dell'interrupt disco) e crea il livello 3 nel quale si esegue la routine di gestione dell'interrupt di clock
- La routine termina l'esecuzione: il kernel recupera il livello di contesto 2 e così via
- Tutti questi passi vengono fatti sempre *all'interno dello stesso processo*: cambia solo la sua parte di contesto dinamica

Algoritmo di gestione delle interruzioni

- L'algoritmo del kernel per la gestione di un'interrupt consiste dei seguenti passi:
 - salva il contesto del processo corrente
 - determina fonte dell'interrupt (trap/interrupt I/O/ ecc)
 - recupera l'indirizzo di partenza della routine di gestione delle interrupt (dal vettore delle interrupt)
 - invoca la routine di gestione dell'interrupt
 - recupera il livello di contesto precedente
- Per motivi di efficienza parte della gestione di interruzioni e trap viene eseguita direttamente dalla CPU (dopo aver seguito un'istruzione): il kernel dipende quindi dal processore

Trap/Interrupt vs Context switch

- Il modo user/kernel permette al kernel di lavorare nel contesto di un altro processo senza dover creare nuovi processi kernel
- Con questo meccanismo un processo in modo kernel può svolgere funzioni logicamente collegate ad altre processi (ad es. la gestione dei dati restituiti da un lettore di disco) e non necessariamente collegate al processo che *ospita* momentaneamente il kernel
- Nota: La gestione di trap/interrupt si basa su una sorta di context switch all'interno di un processo: il controllo non passa ad un'altro processo ma è necessario salvare la parte corrente del contesto dinamico del processo all'interno dello stesso processo

Ma allora quando avviene un context switch tra processi?

- Il kernel vieta context switch arbitrari per mantenere la consistenza delle sue strutture dati
- Il controllo può passare da un processo all'altro in quattro possibili scenari:
 - Quando un processo si sospende
 - Quando termina
 - Quando torna a modo user da una chiamata di sistema ma non è più il processo a più alta priorità
 - Quando torna a modo user dopo che il kernel ha terminato la gestione di un'interrupt a non è più il processo a più alta priorità
- In tutti questi casi il kernel lascia la decisione di quale processo da eseguire allo scheduler