Creació de Processos

Unitat 2.1 - Nel Banqué Torné

---

1. Creació i gestió de processos amb fork

En sistemes operatius **Unix/Linux**, la funció **fork** és fonamental per crear nous processos. Quan un procés utilitza fork, es crea una còpia del procés pare anomenat **procés fill**. Aquesta funció retorna dos valors diferents:

- Al **procés pare**, retorna el **PID** del fill.
- Al **procés fill**, retorna **0**.

1.1 Valors inicials després del fork

- Les **variables locals** en el procés fill són còpies exactes de les del pare en el moment de la crida a fork. Tot i això, el valor retornat per fork és diferent per al pare i el fill.
- Si **fork falla**, no es crea cap procés fill i retorna un valor **negatiu**.
- Després de l'execució de fork, no es pot predir quin dels dos processos (pare o fill) obtindrà la CPU primer, ja que això depèn de la planificació del sistema.

---

2. Sincronització de processos: wait i waitpid

2.1 Funció wait

La funció **wait** suspèn l'execució del procés pare fins que un dels seus fills finalitzi. Això permet la **sincronització** entre processos, evitant que el pare continuï la seva execució fins que els fills hagin acabat.

Exemple de funcionament:

```
printf("My name is King Viserys and I am sad... I have no more child working...\n");
pid_t pid = fork();
if (pid > 0) {
    wait(NULL);  // Espera que el procés fill acabi.
    printf("My child has finished.\n");
}
```

2.2 waitpid

- **waitpid** funciona de manera similar a wait, però permet especificar un **PID** concret d'un fill que volem esperar. Aquesta funció proporciona més control sobre quins processos fills esperem.

Sintaxi:
```
waitpid(pid, &status, 0);
```

- Aquí, pid és el procés fill a esperar i status és una variable on es desa l'estat de sortida del fill.

---

3. Substitució de processos: Funció exec

3.1 Funció exec

La família de funcions **exec** permet substituir el codi d'un procés en execució amb un nou programa. Quan una d'aquestes funcions s'executa, **no es crea un procés nou**, sinó que el procés actual és substituït completament pel nou programa.

Exemple d'ús: Primer, utilitzem **fork** per crear un procés fill, i després aquest fill utilitza **exec** per executar un programa nou:
```
pid_t pid = fork();
if (pid == 0) {
    execlp("/bin/ls", "ls", NULL);  // El procés fill executa 'ls'.
} // Cal posar la ruta, el comande i NULL( perquè s'entengui que acaba aquí)
```

3.2 Importància de exec

Les funcions exec permeten la **reutilització del procés** creat amb fork, substituint el seu contingut amb un altre programa. Aquest mecanisme és fonamental per la càrrega i execució de nous programes en Unix/Linux.

---

4. Processos zombis i orfes

4.1 Processos zombis

Quan un procés fill finalitza, el seu estat es converteix en **EXIT_ZOMBIE** fins que el procés pare realitza una crida a **wait** o **waitpid**. Fins que el pare no llegeixi l'estat de sortida del fill, aquest fill romandrà com a **procés zombi** en el sistema.

- Quan el procés pare fa un wait, el **kernel** elimina el procés zombi de la memòria.

4.2 Processos orfes

Els **processos orfes** són processos fills que han perdut el seu procés pare. A Unix/Linux, aquests processos són adoptats pel procés **init** (PID 1), que es converteix en el seu nou pare. **init** invoca regularment les crides wait per netejar els orfes i evitar que esdevinguin zombis.

---

5. Conclusió

Les funcions **fork**, **wait**, **waitpid** i **exec** són **essencials** per a la gestió de processos en sistemes operatius **Unix/Linux**. Aquestes funcions permeten:

- **Crear** nous processos.
- **Sincronitzar** processos (esperant la seva finalització).
- **Substituir** el codi d'un procés en execució per un programa nou.
- **Gestionar processos zombis** i **orfes**.

La correcta utilització d'aquestes funcions és fonamental per gestionar de manera eficient els processos dins d'un sistema Unix/Linux, permetent una millor assignació de recursos i una execució controlada de programes.