SEMOP

Operazioni sospensive (P e V)

int semop(int semid, struct sembuf *sops, size_t nsops);

- id del pool di semafori di interesse (numero intero)
- Puntatore alla prima delle operazioni da compiere
- Numero di operazioni da compiere

Definizione di un'operazione

```
struct sembuf {
    ushort   sem_num;
    short    sem_op;
    short    sem_flg;
}
```

- Id dello specifico semaforo su cui operare
- Operazione da svolgere (vedere prossimo lucido)
- Opzioni dell'operazione

Supponiamo che val_sem sia il valore attuale del semaforo sem_num-esimo del pool identificato da semid

V

1. Se **(sem_op > 0)** sem_val += sem_op

P

2. Se **(sem_op < 0)** :
   Se | **sem_op** | **≤ sem_val**: sem_val = sem_val - | sem_op |

   Se | **sem_op** | **> sem_val**:
   Se **(sem_flg & IPC_NOWAIT)** è vero semop ritorna subito
   Altrimenti il processo è sospeso finché:
   - qualche altro processo incrementa sem_val di un numero sufficiente di unità
   - il vettore di semafori semid viene rimosso
   - il processo riceve un segnale da catturare

3. Se **(sem_op = = 0)** il processo viene sospeso finché sem_val diventa = = 0

Particolarità di Unix

Attesa del valore 0

Supponiamo di avere N processi il cui codice è suddiviso in parte A e parte B

Tutti i processi devono terminare la parte A prima che anche uno solo possa iniziare la parte B

Possibile implementazione

1. Definiamo un semaforo inizializzato a N
2. Al termine della parte A ciascun processo
   I. decrementa il semaforo di 1
   II. dopodiché si mette in attesa del valore 0
3. Solo quando gli N processi avranno terminato la parte A il semaforo varrà 0 e quindi i processi potranno continuare

Esempio di P e di V realizzate con semop

```
int P (int semid, int semnum) {
   struct sembuf cmd;

   cmd.sem_num = semnum;
   cmd.sem_op = - 1;
   cmd.sem_flg = 0;

   semop(semid, &cmd, 1);
}
```

```
int V (int semid, int semnum) {
   struct sembuf cmd;

   cmd.sem_num = semnum;
   cmd.sem_op = 1;
   cmd.sem_flg = 0;

   semop(semid, &cmd, 1);
}
```