Semafori

`sem_wait(sem);`	Esegue `counter--` e nel caso aggiunge a waiting il processo.
`sem_signal(sem);`	Rilascia un gettone. Esegue `counter++` ed estrae dalla lista dei waiting.
`sem_ini($gettoni);`	Restituisce il primo semaforo libero inserendovi $gettoni gettoni. 0xFFFFFFFF se qualcosa è andato storto.
`sem_valid(id);`	Verifica la validità di un semaforo.

Mutex: $gettone = 1
Sync: $gettone = 0
Nota: tutti gli errori sono 0xFFFFFFFF
Attenzione: Tutti gli id sono `natl`

Processi

`des_p(id);`	Restituisce un des_proc* a partire dall'id.
`c_abort_p();`	Porta all'interruzione del processo. Non effettua return ed è utile nei casi di errore.
`p-> con testo[I_RAX] = $valu e;`	Ritorno della funzione di value.
`rimozione_lista($lista);`	Rimuove la testa dalla lista (max prio)
`inspronti();`	Inserisce **esecuzione** in testa a pronti

Tutti gli `id` sono `natl`

Preemption

```
des_proc* work = rimozione_lista($lista);
inspro nti();
inseri men to_ lis ta( pronti, work);
schedu lat ore();
```

Alla fine ci sarà sicuramente uno tra il processo in esecuzione e work attivo

Utilità

flog(MODE, "msg");	Mostra a schermo il messaggio msg

LOG_DEBUG: costante per debug
LOG_INFO: costante per informazioni
LOG_WARN: costante per avviso

Tab iter

`tab_iter it(tab, begin, dim);`	Uti sc en de de ind
`it.down();`	Pe sc pr ve
`it.is_leaf();`	res `tr` un
`tab_entry& e = it.get_e();`	Re uis rife al tto co
`vaddr v = it.get_v();`	Re uis l'in virt as all in i
`paddr p = extr_I ND_ FIS ICO ($e);`	Re uis l'in fisi ass all
`set_IN D_F ISI CO($e, $paddr);`	Se l'in fisi co de en

`alloca _tab();`	All fra libe des a con una di tra res uen l'in del
`copy_d es( paddr src, paddr dst, natl i, natl n) ;`	Co des da dst pa dal
`set_de s(paddr dst, natl i, natl n, tab_entry e) ;`	Se val su des a p dal 'i'

Utilizzo tab_iter

```
for (tab_iter it(p->cr3, ini_utn_p, dim); it;
it.next()) {
       if (!it.i s_l eaf())
            con tinue;

        tab _en try & e = it.get _e();
        vaddr v = it.get _v();
        paddr p = extr_I ND_ FIS ICO(e); // or set
       /* e &= ~BIT_RW;
              abbiamo modificato RW: invalidamo
             le entrate nel TLB
              inv ali da_ ent rat a_T LB( v);*/
}
```


Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com

Memoria Virtuale

`trasforma(vaddr);`	Restituisce il paddr relativo a un vaddr
`tipo_livello_accesso`	Vedere le note
`alloca_frame();`	restituisce l'indirizzo del frame caricato
`rilascia_frame($paddr);`	Libera il frame in cui è contenuto $paddr
`paddr base = p & ~0xFFF;`	base di un frame
`natl indice = p / DIM_PAGINA;`	Indice di frame
`invali da_ ent rat a_T LB( $va ddr);`	Invalida l'entrata del TLB del processo in esecuzione
cr0	Abilita la pagina-zione
cr1	Non implem-entato
cr2	Indirizzo che ha causato page fault
cr3	paddr albero di traduzione del processo
vdf[];	array dei descrittori di frame
`memcpy( (void) dst, (void) src, int nbytes);`	copia da dst a src nbytes

Funtore

```
class Funtore{
  public:
        paddr pa[DIM _PA GINA];
       natl i;
       // init di i a 0
        Fun tore() : i(0) {}
       // pmap la
        paddr operat or( )(vaddr v){
                return pa[i++];
       }
     // pe unmap
       void operat or( )(v addr, paddr p,
       int){
                pa[i++] = p;
               // oppure trasfo rma(v);
       }
  // attento a ; !!!
```

Elimina da lista

```
des_proc elimina_da_lista(des_proc& testa,
des_proc* p){
  des_proc *prec = &t esta,scorri = testa;
  while (scorri && scorri != p) {
     prec = &s cor ri- >pu nta tore;
     scorri = scorri ->p unt atore;
  }
  if (scorri)
     *prec = scorri ->p unt atore;
  return scorri;
}
```

Rimuove dalla lista puntata da `$testa` il `des_proc* p`.

`map(p- >cr3, vaddr begin, vaddr end, FLAGS, putpaddr);`	Mappa gli indirizzi fisici passati da putpaddr nell'intervallo [begin, end) restituendo il primo che non è riuscito a tradurre
`unmap( p->cr3, vaddr begin, vaddr end, getpaddr);`	Rimuove le traduzioni nell'intervallo [begin, end) da p->cr3

- **tipo**: ini, fin
- **livello**: utn, sis
- **accesso**: p, c

Livello tabelle: 4, 3, 2, 1

FLAGS: bit_RW, bit_US

Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com

I/O

`access (vaddr b, natq dim, bool w, bool s);`	Controlla il **cavallo di Troia** per w(ritable) e s(hared)
`z = input`**`x`**`(reg);`	Inserisce il valore al registro reg in z
`output`**`x`**`(value, reg);`	Inserisce nel registro reg il valore value
`des_x y = new(`**`align_ val _t{dim}`**`) des_x;`	Allinea il nuovo valore in memoria
`natl spostabili = DIM_PAGINA - (paddr & 0xFFF);`	Calcola il numero di byte spostabili per esaurire il frame

x: **b**(yte), **l**(ong), **q**(uad)

Metodi di scorrimento tab_iter

`it.post();` `it;` **`it.nex t_p ost()`**	E u a **p i** V s m u q d **c** u
`tab_iter it(ese cuz ion e->cr3, v, n); it;` **`it.nex t()`**	E u a **a** V s m u q d s c **s** l' p o d q ti

Approccio I/O

inspronti e inserimento_lista(pronti, esecuzione)

La primitiva

```
inspro nti();
```

inserisce il processo in esecuzione **in testa** alla coda pronti.

La primitiva

```
inseri men to_ lis ta( pronti, esecuz ione);
```

inserisce il processo in esecuzione nella coda pronti **in maniera ordinata**. Quindi, se nella coda pronti c'è un processo P1 con **priorità uguale** a quella del processo in esecuzione, quest'ultimo verrà inserito nella lista **dopo** il processo P1.

`inspro nti();` : esecuzione -> P1

`inseri men to_ lis ta( pronti, esecuz ione);` : P1 -> esecuzione

Primitive con tipo di ritorno

- Quando viene chiamata l'interruzione?
- Chi prepara il descrittore?
- Cosa deve fare l'interruzione?
- Come rispondiamo all'interruzione?
- Come ci prepariamo alla prossima interruzione?

Nota: *nei casi, perlomeno di lettura, potrebbe essere sensato impostare i valori passati come parametri dalla primitiva all'interno del descrittore. Sarà poi l'interruzione (se chiamata ogni tot) a sistemare questi valori già inizializzati con i valori aggiornati. Se invece dovesse essere richiamata solo alla fine del processo, sarà compito della read fare tutto il processo e abilitare le interruzioni alla fine facendo riattivare la estern_ce*

Nota2: *quando in una read viene passato un buffer, questo dovrà essere copiato nel descrittore in quanto è il buffer in cui la periferica dovrà scrivere i valori che dovremo leggere.*

Nota3: *se il passaggio di byte tra utente e periferica viene effettuato tramite un buffer di appoggio intermedio, dobbiamo necessariamente usare la memcpy, in quanto questi dati devono essere copiati in una zona di memoria.*

Estern CE

```
// chiamata dalla periferica con un'interruzione
extern " C" void estern _ce (natl id){

       // recuperare periferica
        for (;;){
               // sem_wa it( mutex);
               // libera il sync occupato nella
primitiva
               // sem_si gna l(s ync);
               // sem_si gna l(m utex);
                wfi();
       }
}
```

Se abbiamo una primitiva del tipo

`bool primit iva (pa ram etri)`

dovremo implementarla in due modi diversi nel modulo sistema e nel modulo I/O.

SISTEMA: non possiamo farle restituire un tipo (deve essere void), quindi il valore di ritorno dovremo inserirlo nel contesto del processo in esecuzione.

```
extern " C" void primit iva (pa ram etri) {
//oper azioni
esecuz ion e-> con tes to[ I_RAX] = valore _ri tor
no;
}
```

I/O: possiamo utilizzare i tipi di ritorno delle primitive nel seguente modo:

```
extern " C" bool primit iva (pa ram etri) {
//oper azioni
return valore _ri torno;
}
```


Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
https://apollopad.com