

*** Implementazione di un **server CBS** ***

1 Il server CBS e' essenzialmente un task EDF con un budget caratterizzato da

1.1 Rifornimento immediato non appena si azzera

1.2 Scadenza assoluta (ossia priorita'):

1.2.1 Incrementata del periodo del server se il budget si azzera

1.2.2 Impostata a (ticks+periodo del server) se

1.2.2.1 viene rilasciato un job aperiodico

1.2.2.2 in quel momento il server non e' impegnato

1.2.2.3 il budget corrente e' maggiore o uguale a
(scadenza assoluta-ticks)*(budget massimo/periodo)

1.3 Si vedano i lucidi della lezione R08 per ulteriori dettagli

2 Progetto del server CBS

2.1 Possibilita' di definire diversi server contemporaneamente

2.1.1 Attualmente uno solo

2.2 Scrivere in comm.h la struttura struct cbs_queue che memorizza i job aperiodici che il server esegue

```
+-----+
| struct cbs_queue {
|   struct task *task;
|   int num_workers;
|   job_t workers[MAX_NUM_WORKERS];
|   void *args[MAX_NUM_WORKERS];
|   unsigned int pending[MAX_NUM_WORKERS];
| };
+-----+
```

2.2.1 'task' : puntatore al task EDF che implementa il server CBS

2.2.2 'num_workers' : numero di 'worker' registrati per il server

2.2.3 'workers' : puntatori ai 'worker' registrati

2.2.4 'args' : argomenti da passare alle funzioni

2.2.5 'pending' : contatori di attivazione per i 'worker'

2.2.6 Definire la macro MAX_NUM_WORKERS con valore 8

2.3 Aggiungere il campo 'budget' al descrittore di task

2.3.1 Sempre uguale a zero per un task FPR o semplice EDF

2.3.2 Sempre diverso da zero per un server CBS

2.4 Il campo 'period' di struct task memorizza il periodo del server

2.5 Il campo 'deadline' di struct task memorizza la massima capacita' del server CBS nel senso che "deadline" non la chiamiamo cosi, ma "budget_max", ma sono uguali.

2.5.1 Per leggibilita' usiamo il costrutto del linguaggio C union:

```
+-----+
| struct task {
| [...]
|   unsigned long priority;
|   union {
|     unsigned long deadline;
|     unsigned long budget_max;
|   };
|   unsigned long budget;
| [...]
+-----+
```

2.5.2 Il campo 'deadline' e 'budget_max' condividono lo stesso

spazio di memoria, dunque la union occupa 4 bytes

2.5.3 Il vantaggio e' il poter referenziare quell'area di memoria con un altro nome, aumentando la leggibilita' del codice

2.6 Il campo 'priority' memorizza la scadenza assoluta (EDF)

2.7 Aggiungere il tipo CBS (macro con valore 2) a comm.h

3 Implementazione del server CBS (file cbs.c)

3.1 Allocare una variabile globale cbs0 di tipo struct cbs_queue

3.2 Scrittura della funzione cbs_server(), che implementa il ciclo di esecuzione dei 'worker'

```
+-----+
static void cbs_server(void *arg)
{
    struct cbs_queue *q = (struct cbs_queue *) arg;
    int i;
    for (i=0; i<q->num_workers; ++i)
        if (q->pending[i] > 0)
            break;
    if (i == q->num_workers) {
        puts("\nWARNING: Useless activation of CBS server\n");
        return;
    }
    q->workers[i](q->args[i]);
    irq_disable();
    --q->pending[i];
    irq_enable();                                NB: per task aperiodici, released conta il numero
                                                di job di task aperiodici che sono stati rilasciati.
}
+-----+
```

3.2.1 Alla funzione si passa il puntatore alla struttura cbs_queue

3.2.2 Viene cercato il 'worker' pendente di indice piu' basso

3.2.3 Viene eseguito il 'worker' passandogli l'argomento corrisp.

3.2.4 Viene decrementato il campo pending del worker, disabilitando le interruzioni hardware per evitare race condition

3.2.5 Attenzione: i job aperiodici (worker) non sono schedulati in ordine FIFO, ma in accordo alla priorita' derivata dal loro indice statico

3.3 Scrittura della funzione activate_cbs_worker() per rilasciare un worker al server CBS

3.3.1 Prima versione da eseguire quando le interruzioni sono gia' state disabilitate:

```
+-----+
void irqsafe_activate_cbs_worker(struct cbs_queue *q,
                                  int wid)    worker id
{
    struct task *t = q->task;
    if (wid >= q->num_workers)
        panic0();
    q->pending[wid]++;
    t->released++;                                se c'è un solo job rilasciato e non eseguito, cioè lui stesso
    if (t->released == 1) {                         pd = d(s) * e(s) u32 pd = t->priority * t->budget_max;
        pd = d(s) * e(s) u32 pd = ticks * t->budget_max +
        tdbp = t * e(s) + c(s) * p(s) u32 tdbp = ticks * t->budget_max +
        faccio così per avere istanti di tempo          t->budget * t->period;
        if (time_after_eq(tdbp, pd)) {
            t->priority = ticks + t->period;
}
+-----+
```

```

        trigger_schedule = 1;
        ++globalreleases;
    }
}
+
-----+

```

3.3.2 Seconda versione non 'irq-safe':

```

+-----+
void activate_cbs_worker(struct cbs_queue *q, int wid)
{
    irq_disable();           semplicemente la richiamo bloccando le interruzioni prima
    irqsafe_activate_cbs_worker(q, wid);
    irq_enable();
}
+
-----+

```

3.3.3 Aggiungere il prototipo di activate_cbs_worker() in comm.h

3.3.2 Aggiungere extern di globalreleases e trigger_schedule in comm.h

3.4 Scrittura della funzione decrease_cbs_budget() per decrementare periodicamente il budget

```

+-----+
void decrease_cbs_budget(struct task *t)
{
    t->budget--;
    if (t->budget > 0)
        return;
    t->budget = t->budget_max;
    t->priority += t->period;
    trigger_schedule = 1;
}
+
-----+

```

3.4.1 Aggiungere il prototipo in comm.h

3.5 Modificare la funzione isr_tick() in tick.c:

```

+-----+
++ticks;
if (current->budget > 0)                                <<<
    decrease_cbs_budget(current);                         <<<
check_periodic_tasks();
+
-----+

```

3.5.1 Se il task corrente e' un server CBS, viene invocata la funzione decrease_cbs_budget()

3.6 Scrittura della funzione init_cbs() per inizializzare il server CBS

```

+-----+
int init_cbs(unsigned long max_cap, unsigned long period,
             struct cbs_queue *cbs_q, const char *name)
{
    int tid;
    cbs_q->num_workers = 0;
    data_sync_barrier();
    tid = create_task(cbs_server, cbs_q, period, 10,
                      max_cap, name, CBS);
    if (tid == -1)
        return -1;
    cbs_q->task = taskset + tid;
    data_sync_barrier();
}
+
-----+

```

```

        return 0;
    }
+-----+
3.6.1 create_task() disabilita e riabilita le interruzioni
3.6.2 Utilizziamo data_sync_barrier() per essere certi che il nuovo
      task CBS venga schedulato con i dati validi nel descrittore
      cbs_queue
3.6.3 Il valore '10' per la fase (ritardo del rilascio iniziale del
      primo job) e' sufficientemente lungo per assicurare che
      quando lo scheduler selezionera' il task CBS il valore di
      cbs_q->task sara' stato gia' inizializzato
3.6.4 Per una inizializzazione piu' rigorosa dovremmo modificare
      create_taks() in modo che non riattivi le interruzioni
      all'uscita se in entrata queste erano disabilitate
3.6.5 Aggiungere il prototipo in comm.h

```

3.7 Scrittura della funzione add_cbs_worker() per aggiungere un worker:

```

+-----+
int add_cbs_worker(struct cbs_queue *cbs_q, job_t worker_fn,
                    void *worker_arg)
{
    int i;
    irq_disable();
    i = cbs_q->num_workers;
    if (i >= MAX_NUM_WORKERS) { caso in cui avevamo già raggiunto il massimo numero di worker possibili.
        irq_enable();
        return -1;
    }
    cbs_q->workers[i]      = worker_fn;
    cbs_q->args[i]         = worker_arg;
    cbs_q->pending[i]      = 0;
    cbs_q->num_workers++;
    irq_enable();
    return i;
}
+-----+

```

3.7.1 Aggiungere il prototipo in comm.h

3.8 Modifica della funzione create_task() in tasks.c: aggiungere un
nuovo caso per i server CBS:

```

+-----+
    t->budget = 0;                                <<<
} else if (type == CBS) {                         <<<
    t->priority = 0;                               <<<
    t->budget_max = prio_dead;                   <<<
    t->budget = prio_dead;                      <<<
} else {                                         <<<
    t->budget = 0;                                <<<
+-----+

```

3.8.1 Per distinguere tra un task CBS e gli altri si controlla se
il campo budget e' diverso da zero

3.9 Modifica della funzione check_periodic_tasks() in sched.c:

```

+-----+
if (time_after_eq(now, f->releasetime)) {
    f->releasetime += f->period;
    if (f->budget == 0) { /* not CBS */
        ++f->released;
    }
}
+-----+

```

```

        trigger_schedule = 1;
        ++globalreleases;
    }
}
-----+

```

3.9.1 A differenza di un normale task EDF, il budget del server CBS
non e' ripristinato periodicamente

3.10 Modifica della funzione task_entry_point() in tasks.c

```

-----+
|--t->released;
if (t->deadline != 0 && t->budget == 0) { /* EDF, !CBS */
    if (time_after(ticks, t->priority))
        printf("[%u] EDF task '%s': deadline miss!\n", ticks,
               t->name);
    t->priority += t->period;
}
-----+

```

3.10.1 La deadline del server CBS non viene aggiornata quando un
worker termina essendo aperiodico

4 Test di funzionamento del server CBS

4.1 Scrivere un worker in cbs.c

```

-----+
void test_cbs_job(void *arg)
{
    task all'interno della struct queue è puntatore alla struttura task del server cbs, modellato come un task EDF, cioè periodico.
    associo task a cbs perché cbs fatto come task, e si chiama "cbs_queue" perchè gli associo la coda di job aperiodici.
    struct cbs_queue *q = (struct cbs_queue *) arg;
    struct task *t = q->task;
    static unsigned int count = 0;
    printf("\nCBS: #%-u prio=%u nextrel=%u pending=%u budget=%u\n",
           ++count, t->priority, t->releasetime, q->pending[0],
           t->budget);
}
-----+

```

4.1.1 Aggiungere il prototipo in comm.h

4.2 Aggiungere in __init__() la creazione del server e del worker:

```

-----+
init_taskset();
init_ticks();
if (init_cbs(30, 250, &cbs0, "cbs0") == -1)
    panic0();
if (add_cbs_worker(&cbs0, test_cbs_job, &cbs0) == -1)
    panic0();
-----+

```

4.2.1 Aggiungere in comm.h l'extern per cbs0

4.3 Aggiungere a show_ticks() in main.c l'attivazione per il worker 0
del server CBS

```

-----+
static void show_ticks(void *arg __attribute__ ((unused)))
{
    printf("\nCurrent ticks: %u\n", ticks);
    activate_cbs_worker(&cbs0, 0);
}
-----+

```

4.3.1 Questo e' solo un rapido test: in generale l'indice del
worker e' restituito da add_cbs_worker() ed e' zero solo per

di solito la priorità del worker ci dovrebbe essere ritornata, non dovremmo fornirla noi.

scaletta.txt **Tue Dec 20 15:22:49 2022** **6**

 il primo worker definito

/*
vim: tabstop=4 softtabstop=4 expandtab list colorcolumn=74 tw=73
*/