Trabajo 06 de Ampliación de sistemas operativos — 2007 (Rev. 1)

Fecha: 27/Nov/07

Tercer curso de Ingeniería técnica informática de sistemas

Problema: Pool de procesos

Implementar un pool o conjunto de procesos para atender preticiones. Estas peticiones las generarán diversos procesos con memoria compartida. En este esquema se mantienen un conjunto de procesos que se reutilizan para evitar el coste en latencia de crear y destruir procesos. Además se mantienen un máximo de *N* procesos para evitar degradar la respuesta. Este es un diseño estándar que se puede encontrar en diversos servidores como por ejemplo servidores web.

Cada proceso se encuentra asociado a un nodo de una lista simplemente enlazada circular que se utilizará para irlos asociando a las peticiones que vayan llegando. Hay además un proceso principal. Las peticiones se simularán como una cadena de texto que el proceso principal dejará en el nodo asociado al proceso esclavo. El proceso esclavo se levantará e imprimirá la cadena, simulando así atender la petición.

Cuando un proceso necesite que se atienda una petición y no haya procesos esclavos disponibles, se quedará bloqueado hasta que se libere alguno o creará procesos según las siguientes restricciones.

Los procesos se irán creando hasta llegar a un máximo de N cada vez que el número de procesos esperando a ser atendidos rebasen un número considerado como marca de agua alta. Cuando el número de procesos ocupados baje por debajo de cierto número considerado como marca de agua baja, se liberarán procesos. Un ejemplo de marca de agua alta es que haya el doble de procesos esperando a ser atendidos que procesos esclavos. Un ejemplo de marca de agua baja es que haya la mitad de procesos esclavos ociosos.

Sólo se pueden utilizar como mecanismo de sincronización las funciones *qlock*(2), *rsleep*(2), *rwakeupall*(2) y *rwakeup*(2).

Solución

circlist.h

```
enum{
        /* Process status */
        FREE = 0,        /* Initial state. Keep first */
        BUSY,
        EXIT
};

typedef struct Procnod Procnod;

struct Procnod{
        Rendez  r;      /* should point to general lock */
        Rendez  myr;    /* should point to general lock */
        int             pid;
        int     reqnum;
        int     state;
        void    (*fn)(void);
        Procnod *next;
};

#pragma     varargck    type  "N"   Procnod *

Procnod *       findfreenod(Procnod *n);
Procnod *       newnod(void);
int             Nfmt(Fmt *f);
void            delnod(Procnod *n);
void            insnod(Procnod *n);
void            prlist(Procnod *n);

extern QLock listlck;
extern Procnod *current;
extern int npnods;
extern int nusedpnods;
extern int nwaitingreq;
```

circlist.c

```
#include <u.h>
#include <libc.h>
#include "circlist.h"

QLock   listlck;
Procnod *current;
int             npnods;
int             nusedpnods;
int             nwaitingreq;


static char *stnames[3]={
        [FREE] "free",
        [BUSY] "busy",
        [EXIT] "exiting",
};
```

```
int
Nfmt(Fmt *f)
{
        Procnod *n;

        n = va_arg(f->args, Procnod *);
        if (n == nil)
                return fmtprint(f, "nil");
        return fmtprint(f, "[Me: %p, pid: %d, next: %p, state:%s]",
                        n, n->pid, n->next,stnames[n->state]);
}



Procnod *
newnod(void)
{
        Procnod *p;

        p = mallocz(sizeof(Procnod), 1);
        if (p == nil)
                sysfatal("no memory");
        p->r.l = &listlck;
        p->myr.l = &listlck;
        return p;
}

void
insnod(Procnod *n)
{
        assert(n);
        if(current == nil){
                current = n;
                n->next = n;
        }else{
                n->next = current->next;
                current->next = n;
        }
        npnods++;
}


void
delnod(Procnod *n)
{
        Procnod *c;
```

```
        if(n == nil){
                fprint(2, "delete from empty list\n");
                abort();
        }
        if(n->next == n)
                current = nil;
        else{
                for(c = current; c->next != n; c = c->next)
                        ;
                c->next = n->next;
                if(current == n)
                        current = n->next;
        }
        npnods--;
        free(n);
}


void
prlist(Procnod *n)
{
        Procnod *c;

        if(current != nil)
                print("{cur: %p}", current);
        if(n == nil){
                print("nil\n");
                return;
        }
        print("%N->", n);
        for(c = n->next; c != nil && (c!= n); c = c->next)
                print("%N->", c);
        print("[_]\n");
}


Procnod *
findfreenod(Procnod *n)
{
        Procnod *c;

        if(n == nil)
                return nil;
        if(n->state == FREE)
                return n;
        for(c = n->next; c != nil && (c!= n); c = c->next)
                if(c->state == FREE)
                        return c;
        return nil;
}
```

procattend.c

```
#include <u.h>
#include <libc.h>
#include <thread.h>
#include "circlist.h"
```

```
enum{
        PNODCHUNK = 10,
};

Rendez unservedrdz;


/* internal version without locks
 */
static int
_procrelease(Procnod *n, int dodel)
{
        n->state = FREE;
        n->pid = 0;
        nusedpnods--;

        if(!dodel && nwaitingreq != 0)
                rwakeup(&unservedrdz);

        if(dodel || (nwaitingreq == 0 && nusedpnods <= npnods/2)){
                delnod(n);
                return 1;
        } else
                return 0;
}

void
attendmain(void *v)
{
        Procnod *p;
        int     isfree;

        p = (Procnod *)v;
        for(;;){
                isfree = 0;
                qlock(&listlck);
                while(p->state == FREE) /* wait for work */
                        rsleep(&p->myr);

                p->pid = threadpid(threadid());
                if(p->state == EXIT)
                        isfree = _procrelease(p, 1);
                qunlock(&listlck);
                if(isfree)
                        threadexits(nil);
                // print("process %d, attending req %d\n", p->pid, p->reqnum);
                (*p->fn)();
                qlock(&listlck);
                isfree = _procrelease(p, 0);
                qunlock(&listlck);
                if(isfree)
                        exits(nil);
        }
}
```

```
void
createpnods(int np)
{
        int i;
        Procnod *p;

        for(i = 0; i < np; i++){
                qlock(&listlck);
                p = newnod();   //and its proccreate
                p->pid = threadpid(threadid());
                proccreate(attendmain, p, 8*1024);
                insnod(p);
                qunlock(&listlck);
        }
}

int
initprocs(int nprocs)
{
        qlock(&listlck);
        assert(!npnods);
        unservedrdz.l = &listlck;
        qunlock(&listlck);

        createpnods(nprocs);

        return 0;
}


void
prnods(void)
{
        qlock(&listlck);
        prlist(current);
        qunlock(&listlck);
}

int
procrelease(Procnod *n)
{
        qlock(&listlck);
        fprint(2, "warning: killing node %N and not exiting", n);
        _procrelease(n, 1);
        qunlock(&listlck);
        return 0;
}

int
reqattend(void (*fn)(void), int reqnum)
{
        Procnod *n;
```

```
        qlock(&listlck);
        nwaitingreq++;
        //print("attending request: nused: %d, npnods: %d, nwaitingreq: %d\n",
        //      nusedpnods, npnods, nwaitingreq);
        //prlist(current);
        assert(nusedpnods <= npnods);

        if(npnods  <= nwaitingreq/2){
                createpnods(PNODCHUNK);
                rwakeupall( &unservedrdz );     /* whip the new slaves */
        }
        while( npnods <= nusedpnods )
                rsleep( &unservedrdz );

        nwaitingreq--;
        //print("woke up ready to rock--\n");
        //print("attending request: nused: %d, npnods: %d, nwaitingreq: %d\n",
        //      nusedpnods, npnods, nwaitingreq);
        //prlist(current);
        n = findfreenod(current);
        assert(n);
        current=n->next;
        nusedpnods++;
        n->fn = fn;
        n->reqnum = reqnum;
        n->state = BUSY;
        rwakeup(&n->myr);
        qunlock(&listlck);

        return 0;
}



void
exitall(void)
{
        int exit;

        exit = 0;
        do{
                qlock(&listlck);
                if(current && current->state == FREE){
                        current->state = EXIT;
                        rwakeup(&current->myr);
                }
                if(!current && !nwaitingreq)
                        exit = 1;
                qunlock(&listlck);
        }while(exit == 0);
}

int counter;
QLock lckctr;
```

```
void
func(void){
        int ctr;
        qlock(&lckctr);
        print("The counter is %d\n", counter++);
        ctr = counter;
        qunlock(&lckctr);
        sleep(1000);
        print("exiting request---- %d\n", ctr);
}

void
threadmain(int , char **)
{
        fmtinstall('N', Nfmt);
        initprocs(5);
        prnods();
        reqattend(func, 12);

        reqattend(func, 5);

        reqattend(func, 13);

        reqattend(func, 8);

        reqattend(func, 17);

        reqattend(func, 19);

        prnods();

        print("going to exit\n");
        exitall();
        prnods();

        print("exiting\n");
}
```

Se ha separado la lista circular en otro fichero para facilitar su depuración. Además, varias sentencias utilizadas para depuración se han dejado comentadas, para ilustrar un poco mejor el código.

—