```
/*********************************************************************
        Code for power supply task controller that runs at 100 Hz.

        ALS Controls Upgrade
        Susanna Jacobson
        LBNL

********************************************************************* /

#include <vxWorks.h>
#include <stdio.h>
#include <sysLib.h>
#include <taskLib.h>
#include <tickLib.h>

#include "PSCTasker.h"

#define MAX_TASKER_ENTRIES 16

ULONG   PSCticks_at_start;
ULONG   PSCticks_at_end;
ULONG   PSCtasker_ticks;

struct tasker_entry {
        void    (*func)(void *);        /* Pointer to function to execute * /
        void    *parg;                  /* Call function with this arg ptr * /
        char    *func_name;             /* Function name for diagnostics * /
        char    *arg_name;              /* Argument name for diagnostics * /
};

int     PSCnum_tasks = 0;
struct tasker_entry PSCtasks[MAX_TASKER_ENTRIES];

BOOL    PSCstop = FALSE;

/*********************************************************************
        PSCRegister: add a function to be called and its argument to
                the tasker's table. Returns error code if function
                pointer is invalid or if table is already full.
 ***    Note that multiple calls of this function must occur        ***
 ***            within a single thread. There is no locking         ***
 ***            or other provision in this function for             ***
 ***            multi-thread operation.                             ***
********************************************************************* /
int
PSCRegister(void (*func)(void *), void *parg, char *fname, char *argname)
{
        char    *name = "(noname)", *argnm = "(noargname)";

        if (fname != (char *)NULL) name = fname;
        if (argname != (char *)NULL) argnm = argname;

        if (func == (void(*)(void *))NULL) {
                printf(
"PSCRegister: NULL function pointer for function %s \n\
        with arg at %s (0x%08x).\n", name, argnm, (UINT)parg);
                return PSCerr_null_func_ptr;
        }
/* It's OK if parg is NULL. * /
        if (PSCnum_tasks == MAX_TASKER_ENTRIES) {
                printf(
"PSCRegister: Table full, cannot add function %s() (0x%08x)\n\
        with arg at %s (0x%08x).\n", name, (UINT)func, argnm, (UINT)parg);
                return PSCerr_table_full;
```

PSCRegister

...PSCRegister

```
        }
        PSCtasks[PSCnum_tasks].func = func;
        PSCtasks[PSCnum_tasks].parg = parg;
        PSCtasks[PSCnum_tasks].func_name = name;
        PSCtasks[PSCnum_tasks].arg_name = argnm;
        PSCnum_tasks++;
        return OK;
}
```

PSCTasker

```
void
PSCTasker(void)
{
        int     i;

        /* Initialization:
        *       Enable loop operation, then
        *       get PSCticks_at_start and set PSCtasker_ticks to same.
        */
        PSCstop = FALSE;

        PSCticks_at_start = tickGet();
        PSCtasker_ticks = PSCticks_at_start;

        /* The main loop:
        *       If there are functions in the table,
        *               step through table calling functions.
        *       Wait one tick, then increment tick counter.
        */
        while(!PSCstop) {
                if (PSCnum_tasks > 0) {
                        /* Note that parg is NOT dereferenced. */
                        for (i=0; i<PSCnum_tasks; i++){
                                (*PSCtasks[i].func)(PSCtasks[i].parg);
                        }
                }
                taskDelay(1);
                PSCtasker_ticks++;
        }

        /* Exits only if PSCstop set FALSE (0).
        *       Print diagnostic for comparing tick counts.
        */
        PSCticks_at_end = tickGet();
        printf(
"PSCTasker exit.  Tick counts: System %d, Tasker %d.\n\
        started %d; difference %d tick(s) in %d ticks total\n",
                (UINT)PSCticks_at_end, (UINT)PSCtasker_ticks,
                (UINT)PSCticks_at_start,
                (UINT)(PSCticks_at_end-PSCtasker_ticks),
                (UINT)(PSCticks_at_end-PSCticks_at_start));
}
```

PSCTaskerStop

```
void
PSCTaskerStop(void)
{
        PSCstop = TRUE;
}


/*****************************************************************************
        PSCTester: Just runs for specified number of seconds and then
                tells PSCTasker to exit.
        Use:
                -> sp PSCTasker
```

```
                        –> sp PSCTester(10)          # 10 seconds in this example.
************************************************************************ /
void
PSCTester(int secs)
{
        if (secs >= 1) {
                taskDelay(secs*sysClkRateGet());
        }
        else {
                printf("PSCRateCheck: number of seconds must be 1 or more\n");
        }
        PSCstop = TRUE;
        exit(0);
}
```

PSCTester

```
void
PSCRateCheck(int secs, int d)
{
        int i, j, hz, ticks;

        if (d < 0) ticks = 1;
        else ticks = d;

        if (secs >= 1) {
                printf("\n");
                hz = sysClkRateGet();
                PSCticks_at_start = tickGet();
                for (i=0; i<secs; i++){
                        for(j=0; j<hz; j++) {
                                taskDelay(ticks);
                        }
                        printf(".");
                }
                PSCticks_at_end = tickGet();
                printf("\nPSCRateCheck: %d seconds = %d ticks\n", secs,
                        (int)(PSCticks_at_end–PSCticks_at_start));
        }
        else {
                printf("PSCRateCheck: number of seconds must be 1 or more\n");
        }

}
```

PSCRateCheck

```
void
PSCTaskerTablePrint(void)
{
        int             i;
        struct tasker_entry
                        *p = PSCtasks;

        for (i=0; i<PSCnum_tasks; i++) {
                printf("Entry %d: fptr = 0x%08lx (%s) w/ arg at 0x%08lx (%s)\n",
                        i, (ULONG)p–>func, p–>func_name,
                        (ULONG)p–>parg, p–>arg_name);
                p++;
        }
}
```

PSCTaskerTablePrint