Operating System Concepts

We implement process identifiers and the explicit recording of process states by making the following changes to the files indicated.

- in procP.h: new datatype for process states and two new fields in the process control block

```c
typedef enum {
    RUNNING,
    READY,
    SUSPENDED,
    DELAYED
} state;
```

- in proc.h: the create function now returns an int that gives the process id

```c
int create(void (*func)(void), int stack, int priority);
```

- in proc.c: amended implementation of create, dispatch and setready function plus the change_state function giving diagnostic output

```c
int create(void (*func)(void), int stack, int pr)
```
newpcb->next = 0;
*/
setready( newpcb );
EXITKERNEL
return newpcb->pid;
}

void dispatch(void) {
    process oldrunning;
    int pr;
    oldrunning = running;
    pr = 3;
    while (isEmpty(&readyQ[pr])) pr--;
    running = getHead(&readyQ[pr]);
    int_clear();
    /* change process state */
    change_state (running, RUNNING);
    pswitch(&oldrunning->context, &running->context);
}

void setready(process pcb) {
    if ( pcb == 0 ) {
        return;
    } /* record state change */
    change_state(pcb, READY);
    addTail(&readyQ[pcb->pr], pcb);
}

void print_state (state s) {
    switch (s) {
        case RUNNING : con_outs("rn"); break;
        case READY : con_outs("rdy"); break;
        case SUSPENDED : con_outs("ssp"); break;
        case DELAYED : con_outs("dld"); break;
    }
}

void print_pid (process p) {
    if (p != 0) { con_outdec(p->pid); }
}

/* new change_state function */
void change_state( process p, state new_s ) {
    if ( p != 0 ) {
        /* for verification: */
        con_outs("["); print_pid(p);
        con_outs(" ");
        print_state(p->s); con_outs(" ---> ");
        /* assignment */
        p->s = new_s;
        print_state(p->s); con_outs("]");
    }
}

• in sem.c: record the state change in the P-operation (V does this automatically via setReady)

void P(semaphore * s) {
    ENTERKERNEL
    if (s->value) {
        s->value--;
    } else {
        addTail(s->waiting, running);
        /* record the new process state */
        change_state(running, SUSPENDED);
        dispatch();
    }
    EXITKERNEL
}

• in time.c: record the state change in delay; again this is done automatically for processes woken up via setReady.

void delay(int time)

delay(1000);
process p0, p1;

ENTERKERNEL

if (time > 0) {
    if (delayQ == 0) {
        delayQ = running;
        delayQ->next = 0;
    } else {
        p0 = p1 = delayQ;
        for (;;) {
            if (!p1)
                break;
            if (time > p1->pdelay) {
                time -= p1->pdelay;
                p0 = p1;
                p1 = p1->next;
            } else
                break;
        }
    }
    running->next = p1;
    if (p0 == p1)
        delayQ = running;
    else
        p0->next = running;
    if (p1)
        p1->pdelay -= time;
    running->pdelay = time;
    /* record state change */
    change_state (running, DELAYED);
    dispatch();
}

EXITKERNEL