SE 421 Fall 2022 Homework 2 (30 points), Assigned: 9/30, due: Friday, 10/7

Name (Last, First): Duberstein, Wyatt

You are given the complete XINU code. The line numbers are as in the included `dskenq`, `dskqopt`, and `dsinter` codes.

Submission: (a) The answers should be typed. (b) The first page should include the top two lines with your last and the first name. (c) Include the questions followed by answers.

Name the submission file HW2-lastname-firstname.

This homework will be discussed in class on Friday, 9/30. Listen to the lecture before asking questions. The last two MCQs will help you. Review them.

The submission and late policy are as described in the syllabus (see the section **Homework and Project Submission Policy)**

.

1. (6 points) Fill the following table by applying the pattern (discussed in the class) to determine the feasibility of execution sequences in `dskqopt` for each of its three callers: `dsread`, `dswrite`, and `dsksync`

	Feasible or not when called from		
Governing Conditions in dskqopt	**dsread**	**dswrite**	**dsksync**
drptr->drop==DSYNC || (drptr->drop==DREAD && p->drop==DREAD)	Yes	No	Yes
drptr->drop == DSEEK	No	No	No
p->drop == DSEEK	Yes	Yes	Yes
p->drop==DWRITE && drptr->drop==DWRITE	No	Yes	No
drptr->drop==DREAD && p->drop==DWRITE	Yes	No	No
drptr->drop==DWRITE && p->drop==DREAD)	No	No	No

2. (9 points) In `dskenq` and `dskqopt`, the `drptr` is aliased and it is accessible using `devptr`, a pointer to a structure of type `devsw` which is declared in one of the header files included in `dswrite`. Answer the following:

   a. Which header file declares the structure devsw? (1 point)

      conf.h

   b. Randomly check ten *.c files. List the ten files you have checked. For each of them, indicate if the file contains `conf.h`? (3 points)

      a. dgmcntl.c = Yes
      b. dgmopen.c = Yes
      c. dgread.c = Yes
      d. dot2ip.c = Yes
      e. dsinter.c = Yes
      f. dskbcpy.c = Yes
      g. dgparse.c = Yes
      h. dsopen.c = Yes
      i. dsread.c = Yes
      j. dsseek.c = Yes

c. *Find five files* (do not include `dsinter.c`) that include `conf.h` and a `freebuf` call. List these files. For each of them give the number of `freebuf` calls. (3 points)

rarp_in.c = 1 call
ip_in.c = 2 calls
icmp_in.c = 2 calls
ibget.c = 1 call
ethinter.c = 2 calls

d. Of the *five files you found*, does any one of them include a `freebuf` that frees memory using `drptr`, a pointer to a structure of the type `dreq` - Yes or No? If yes, give the name of the file. (2 points)

No. I did find one that I didn't list that did that, dsread.c.

3. (11 points) The file `dsinter.c` includes `conf.h` and a `freebuf` call using `drptr`. Answer the following questions for `dsinter.c`.

a. How many execution sequences are there? (2 points)

40

b. List all the execution subsequences starting at line 27. (3 points)

27,28,29,31,33,34
27,28,29,31,32,33,34
27,28,29,31,32,36,37,40,42
27,28,29,31,32,36,39,40,42
27,28,29,31,32,36,39,42
27,29,31,33,34
27,29,31,32,33,34
27,29,31,32,36,37,40,42
27,29,31,32,36,39,40,42
27,29,31,32,36,39,42

c. Is 17, 18, 19(F), 23(F), 26, 27(F), 29, 31, 32, 33, 34 a feasible execution sequence for analyzing the memory leak for dswrite – yes, or no? If yes, justify in one sentence. (2 points)

No, because the freebuf methods are never used so you can't experience the behavior of those methods

d. Is 17, 18, 19(F), 23(F), 26, 27(F), 29, 31, 32, 36, 37, 39, 42 a feasible execution sequence for analyzing the memory leak for `dswrite` – yes, or no? If yes, justify in one sentence. (2 points)

Yes, since the memory is freed then it can be analyzed

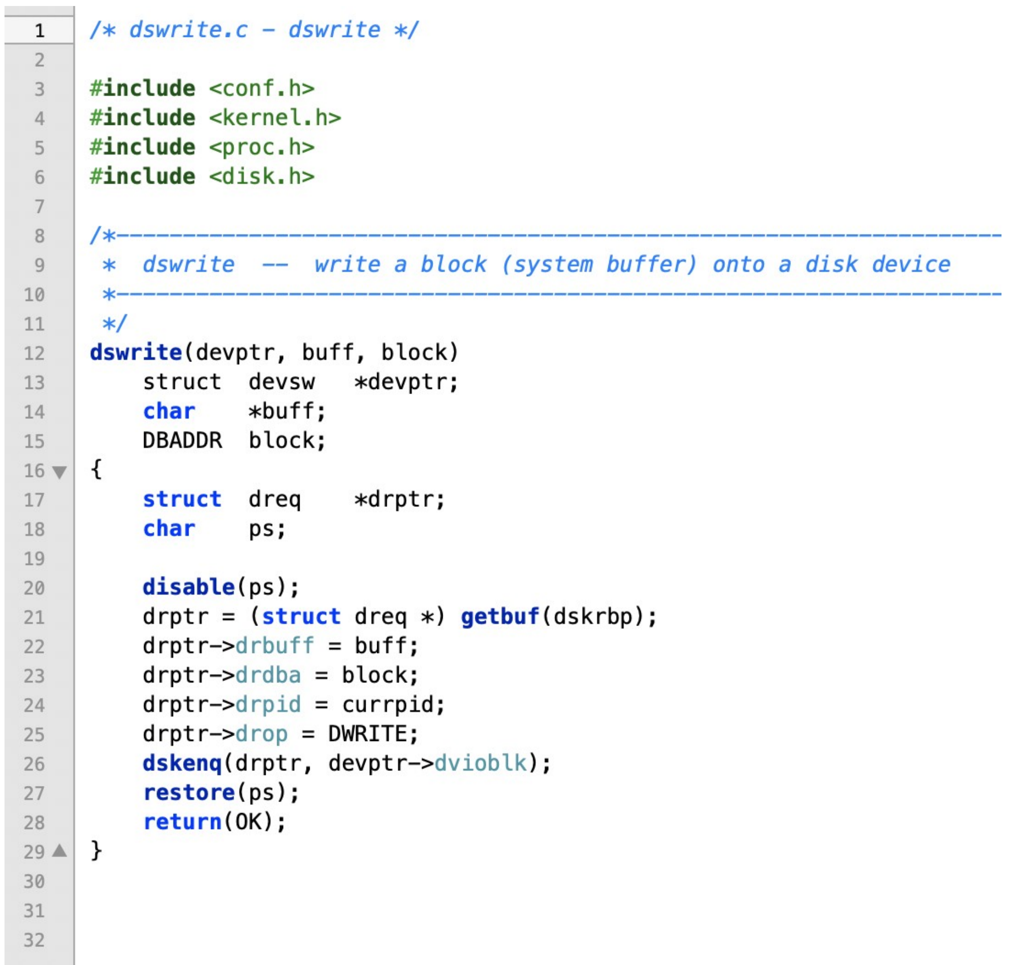
e. Does `dsinter` free the memory allocated in `dswrite` – yes, or no? If yes, give the relevant line number. (2 points)

Yes, line 40

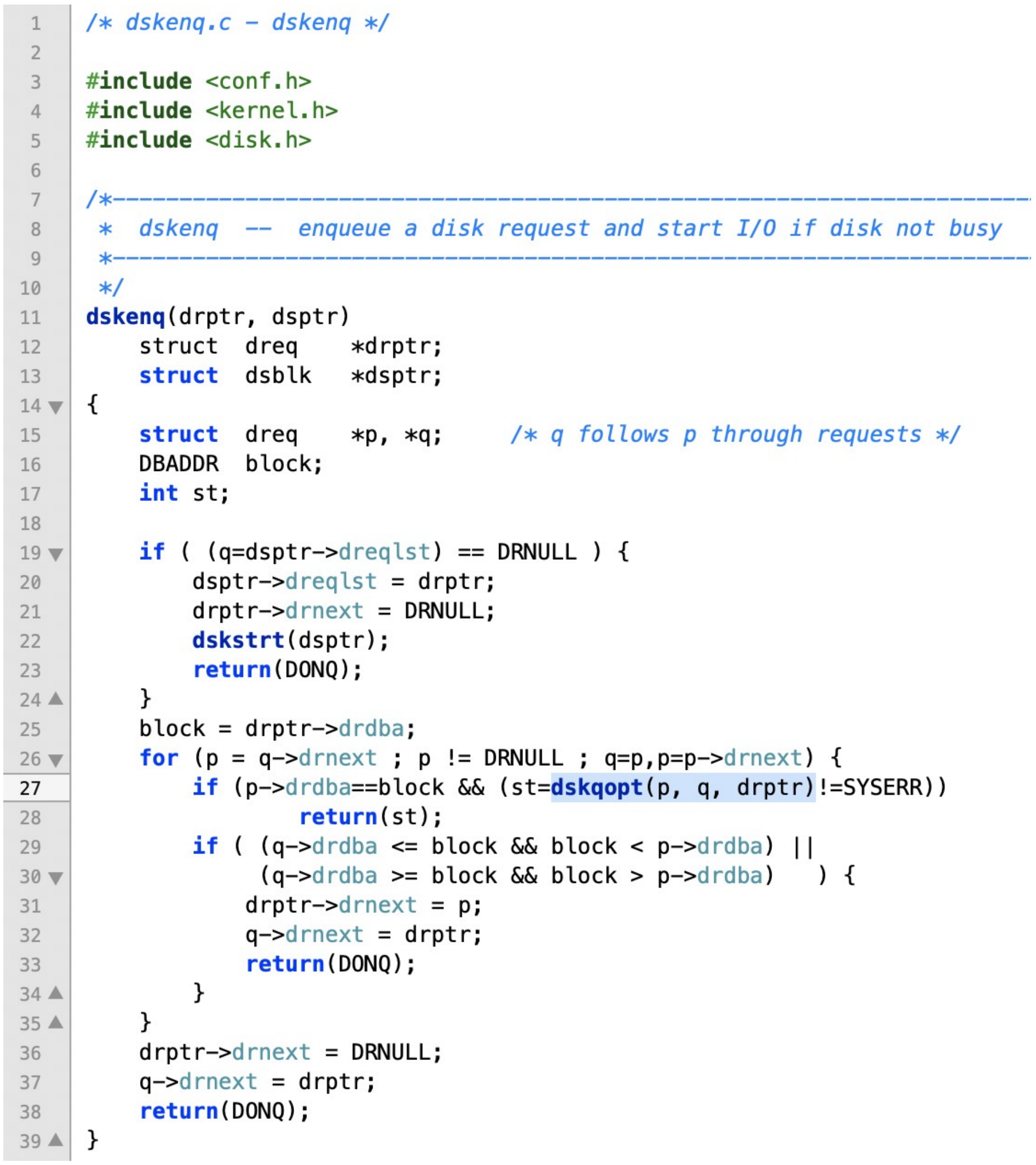
4. (4 points) Answer True or False for each the following questions to summarize the memory leak analysis for the allocation in `dswrite`.

a. The feasible `dskenq` execution sequences have two possibilities: (a) `drptr` is aliased and it is accessible through `devptr` or `dsptr`, (b) aliased `drptr` is passed to `dskqopt`.
    a. True
b. The feasible `dskqopt` execution sequences have two possibilities: (a) `drptr` is aliased and it is accessible through `devptr` or `dsptr`, (b) the memory is deallocated by `freebuf(drptr)`.
    a. False
c. The feasible `dsinter` execution sequences have two possibilities: (a) a *panic* alert, (b) the memory is deallocated by `freebuf(drptr)`.
    a. False
d. Except the case of *panic* alert, the allocated memory allocated is always deallocated.
    a. False

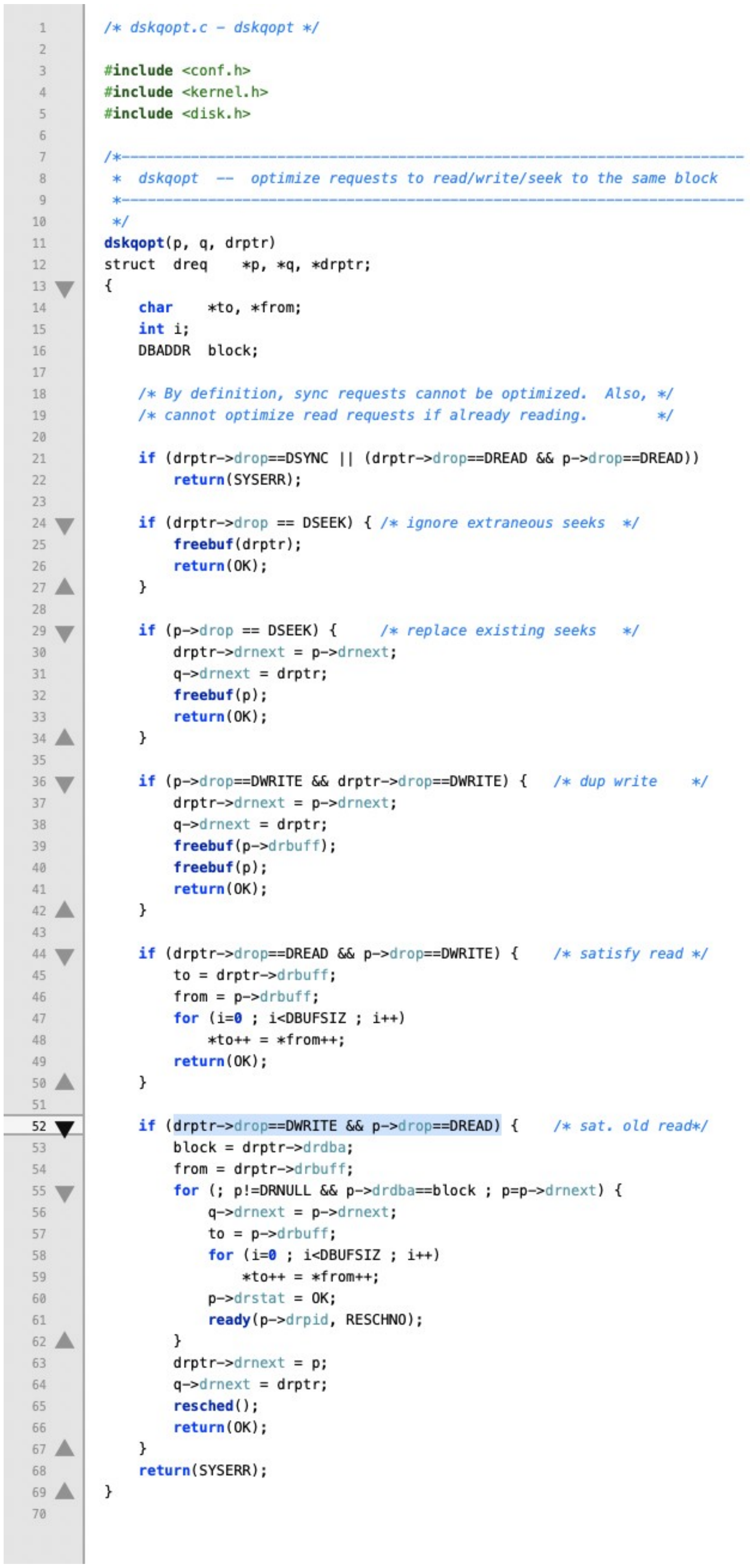
Use the line numbers from the following `dswrite`, `dskenq`, `dskqopt`, and `dsinter` codes



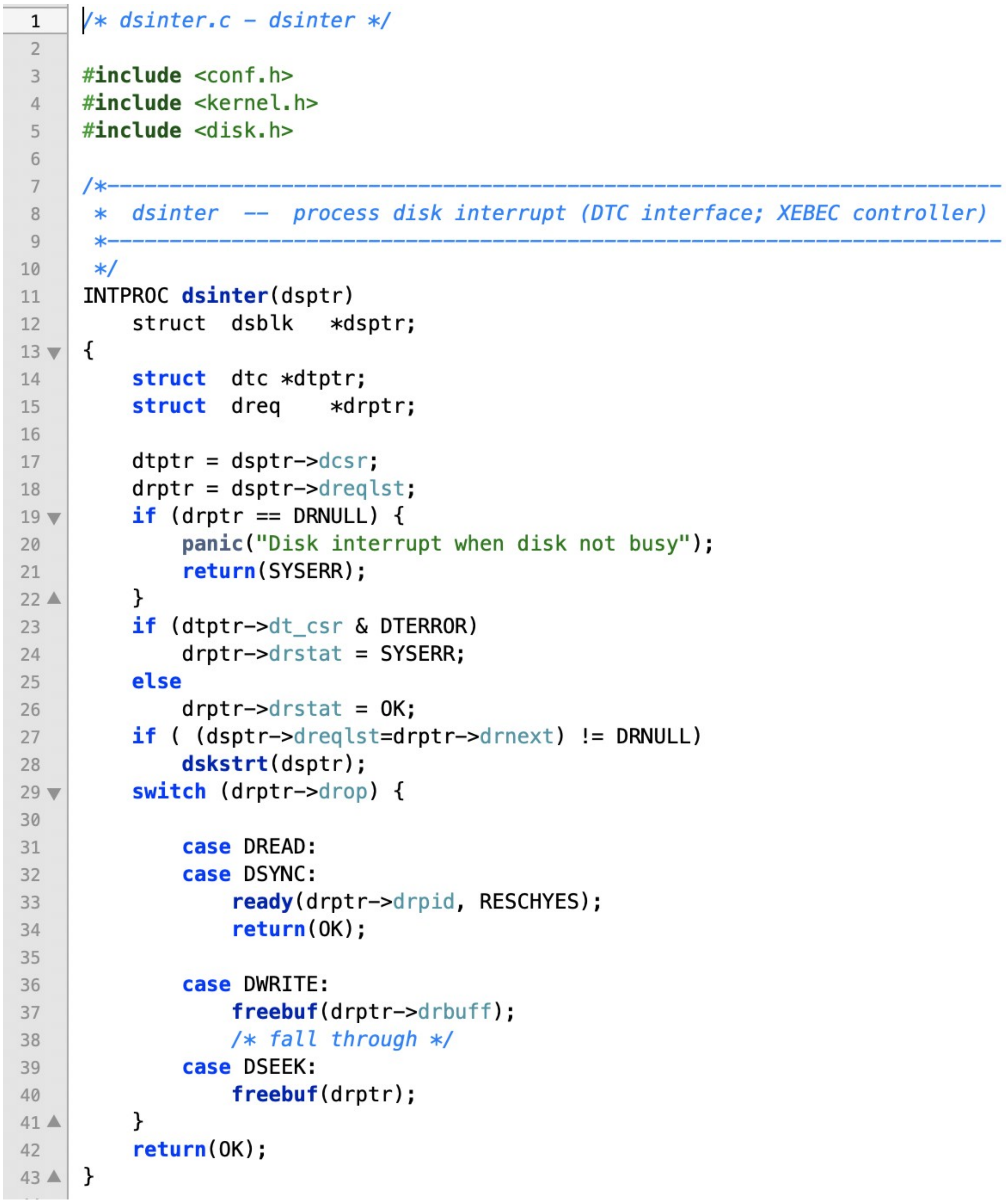
```
1   /* dswrite.c - dswrite */
2
3   #include <conf.h>
4   #include <kernel.h>
5   #include <proc.h>
6   #include <disk.h>
7
8   /*------------------------------------------------------------------------
9    *  dswrite  --  write a block (system buffer) onto a disk device
10   *------------------------------------------------------------------------
11   */
12  dswrite(devptr, buff, block)
13      struct  devsw   *devptr;
14      char    *buff;
15      DBADDR  block;
16  {
17      struct  dreq    *drptr;
18      char    ps;
19
20      disable(ps);
21      drptr = (struct dreq *) getbuf(dskrbp);
22      drptr->drbuff = buff;
23      drptr->drdba = block;
24      drptr->drpid = currpid;
25      drptr->drop = DWRITE;
26      dskenq(drptr, devptr->dvioblk);
27      restore(ps);
28      return(OK);
29  }
30
31
32
```



```c
/* dskenq.c - dskenq */

#include <conf.h>
#include <kernel.h>
#include <disk.h>

/*------------------------------------------------------------------------
 *  dskenq  --  enqueue a disk request and start I/O if disk not busy
 *------------------------------------------------------------------------
 */
dskenq(drptr, dsptr)
	struct	dreq	*drptr;
	struct	dsblk	*dsptr;
{
	struct	dreq	*p, *q;		/* q follows p through requests */
	DBADDR	block;
	int st;

	if ( (q=dsptr->dreqlst) == DRNULL ) {
		dsptr->dreqlst = drptr;
		drptr->drnext = DRNULL;
		dskstrt(dsptr);
		return(DONQ);
	}
	block = drptr->drdba;
	for (p = q->drnext ; p != DRNULL ; q=p,p=p->drnext) {
		if (p->drdba==block && (st=dskqopt(p, q, drptr)!=SYSERR))
				return(st);
		if ( (q->drdba <= block && block < p->drdba) ||
		     (q->drdba >= block && block > p->drdba)    ) {
			drptr->drnext = p;
			q->drnext = drptr;
			return(DONQ);
		}
	}
	drptr->drnext = DRNULL;
	q->drnext = drptr;
	return(DONQ);
}
```



```c
/* dskqopt.c - dskqopt */

#include <conf.h>
#include <kernel.h>
#include <disk.h>

/*------------------------------------------------------------------------
 *  dskqopt  --  optimize requests to read/write/seek to the same block
 *------------------------------------------------------------------------
 */
dskqopt(p, q, drptr)
struct  dreq    *p, *q, *drptr;
{
    char    *to, *from;
    int i;
    DBADDR  block;

    /* By definition, sync requests cannot be optimized.  Also, */
    /* cannot optimize read requests if already reading.        */

    if (drptr->drop==DSYNC || (drptr->drop==DREAD && p->drop==DREAD))
        return(SYSERR);

    if (drptr->drop == DSEEK) { /* ignore extraneous seeks  */
        freebuf(drptr);
        return(OK);
    }

    if (p->drop == DSEEK) {     /* replace existing seeks   */
        drptr->drnext = p->drnext;
        q->drnext = drptr;
        freebuf(p);
        return(OK);
    }

    if (p->drop==DWRITE && drptr->drop==DWRITE) {   /* dup write    */
        drptr->drnext = p->drnext;
        q->drnext = drptr;
        freebuf(p->drbuff);
        freebuf(p);
        return(OK);
    }

    if (drptr->drop==DREAD && p->drop==DWRITE) {    /* satisfy read */
        to = drptr->drbuff;
        from = p->drbuff;
        for (i=0 ; i<DBUFSIZ ; i++)
            *to++ = *from++;
        return(OK);
    }

    if (drptr->drop==DWRITE && p->drop==DREAD) {    /* sat. old read*/
        block = drptr->drdba;
        from = drptr->drbuff;
        for (; p!=DRNULL && p->drdba==block ; p=p->drnext) {
            q->drnext = p->drnext;
            to = p->drbuff;
            for (i=0 ; i<DBUFSIZ ; i++)
                *to++ = *from++;
            p->drstat = OK;
            ready(p->drpid, RESCHNO);
        }
        drptr->drnext = p;
        q->drnext = drptr;
        resched();
        return(OK);
    }
    return(SYSERR);
}
```



```c
/* dsinter.c - dsinter */

#include <conf.h>
#include <kernel.h>
#include <disk.h>

/*------------------------------------------------------------------------
 *  dsinter  --  process disk interrupt (DTC interface; XEBEC controller)
 *------------------------------------------------------------------------
 */
INTPROC dsinter(dsptr)
    struct  dsblk   *dsptr;
{
    struct  dtc *dtptr;
    struct  dreq    *drptr;

    dtptr = dsptr->dcsr;
    drptr = dsptr->dreqlst;
    if (drptr == DRNULL) {
        panic("Disk interrupt when disk not busy");
        return(SYSERR);
    }
    if (dtptr->dt_csr & DTERROR)
        drptr->drstat = SYSERR;
    else
        drptr->drstat = OK;
    if ( (dsptr->dreqlst=drptr->drnext) != DRNULL)
        dskstrt(dsptr);
    switch (drptr->drop) {

        case DREAD:
        case DSYNC:
            ready(drptr->drpid, RESCHYES);
            return(OK);

        case DWRITE:
            freebuf(drptr->drbuff);
            /* fall through */
        case DSEEK:
            freebuf(drptr);
    }
    return(OK);
}
```