# Operating Systems

High-Level Memory Management
in Xinu



## High-Level Memory Management

- Distinct from the lower-level getmem() freemem() interfaces
  - Provides basic memory management
  - Explicit memory allocation and release allow memory to be exhausted
- Higher-level memory management can be employed to divide resources more fairly
- A motivating example is network-centric applications that send or receive messages



## Buffer Pools

- Buffer pools are a common design pattern
- Divide memory into a set of pools
- Each pool contains a fixed number of blocks of the same size, much like a slab allocator
  - Buffer reflects the intended use for I/O
- Each buffer pool is identified by an integer, a buffer pool ID
- Buffer allocation is synchronous in that a process requesting a buffer will be blocked until the request can be satisfied



## Buffer Pools

```
/* bufpool.h */

#ifndef NBPOOLS
#define NBPOOLS 20      /* Maximum number of buffer pools */
#endif

#ifndef BP_MAXB
#define BP_MAXB 8192        /* Maximum buffer size in bytes       */
#endif

#define BP_MINB 8       /* Minimum buffer size in bytes       */
#ifndef BP_MAXN
#define BP_MAXN 2048        /* Maximum number of buffers in a pool    */
#endif
```

```
struct  bpentry {       /* Description of a single buffer pool   */
   struct   bpentry *bpnext;/* pointer to next free buffer       */
   sid32    bpsem;      /* semaphore that counts buffers  */
                        /*    currently available in the pool */
   uint32   bpsize;     /* size of buffers in this pool       */
   };

extern  struct  bpentry buftab[];/* Buffer pool table         */
extern  bpid32  nbpools;    /* current number of allocated pools */
```

5

# Allocating a Buffer

```
/* getbuf.c - getbuf, nbgetbuf */

#include <xinu.h>

/*------------------------------------------------------------------------
 *  getbuf  --  get a buffer from a preestablished buffer pool
 *------------------------------------------------------------------------
 */
char    *getbuf(
          bpid32        poolid        /* index of pool in buftab       */
        )
{
   intmask  mask;           /* saved interrupt mask       */
   struct   bpentry *bpptr;     /* pointer to entry in buftab */
   struct   bpentry *bufptr;    /* pointer to a buffer     */
```

6

```
   mask = disable();

   /* Check arguments */

   if ( (poolid < 0  ||  poolid >= nbpools) ) {
    restore(mask);
    return (char *)SYSERR;

   }
   bpptr = &buftab[poolid];

   /* Wait for pool to have > 0 buffers and allocate a buffer */

   wait(bpptr->bpsem);
   bufptr = bpptr->bpnext;

   /* Unlink buffer from pool */

   bpptr->bpnext = bufptr->bpnext;

   /* Record pool ID in first four bytes of buffer   and skip */
```

7

```
   *(bpid32 *)bufptr = poolid;
   bufptr = (struct bpentry *)(sizeof(bpid32) + (char *)bufptr);
   restore(mask);
   return (char *)bufptr;
}

/*------------------------------------------------------------------------
 * nbgetbuf - a non-blocking version of getbuf
 *------------------------------------------------------------------------
 */
char    *nbgetbuf(
          bpid32        poolid          /* index of pool in buftab       */
   )
{
   intmask  mask;
   char *buf;

   mask = disable();

   /* Check arguments */
```

8

```
if ( (poolid < 0  ||  poolid >= nbpools) ) {
    restore(mask);
    return (char *)SYSERR;

   }

   /* If the call will block, return an error */

   if (semcount(buftab[poolid].bpsem) <= 0) {
    restore(mask);
    return (char *)SYSERR;
   }
   buf = getbuf(poolid);
   restore(mask);
   return buf;
}
```

9

# Returning Buffers To The Buffer Pool

```
/* freebuf.c - freebuf */

#include <xinu.h>

/*------------------------------------------------------------------
 * freebuf - free a buffer that was allocated from a pool by getbuf
 *------------------------------------------------------------------
 */
syscall freebuf(
     char      *bufaddr   /* address of buffer to return    */
   )
{
   intmask  mask;          /* saved interrupt mask       */
   struct   bpentry *bpptr;    /* pointer to entry in buftab */
   bpid32   poolid;        /* ID of buffer's pool    */
```

10

```
   mask = disable();

   /* Extract pool ID from integer prior to buffer address */

   bufaddr -= sizeof(bpid32);
   poolid = *(bpid32 *)bufaddr;
   if (poolid < 0  ||  poolid >= nbpools) {
    restore(mask);
    return SYSERR;
   }

   /* Get address of correct pool entry in table */

   bpptr = &buftab[poolid];

   /* Insert buffer into list and signal semaphore */

   ((struct bpentry *)bufaddr)->bpnext = bpptr->bpnext;
   bpptr->bpnext = (struct bpentry *)bufaddr;
   signal(bpptr->bpsem);
   restore(mask);
   return OK;
}
```

11

# Creating a Buffer Pool

```
/* mkbufpool.c - mkbufpool */

#include <xinu.h>

/*------------------------------------------------------------------
 *  mkbufpool - allocate memory for a buffer pool and link the
   buffers
 *------------------------------------------------------------------
 */
bpid32  mkbufpool(
     int32     bufsiz,    /* size of a buffer in the pool   */
     int32     numbufs    /* number of buffers in the pool*/
   )
{
   intmask  mask;          /* saved interrupt mask       */
   bpid32   poolid;        /* ID of pool that is created */
   struct   bpentry *bpptr;    /* pointer to entry in buftab */
```

12

```
char    *buf;          /* pointer to memory for buffer   */

   mask = disable();
   if (bufsiz<BP_MINB || bufsiz>BP_MAXB
       || numbufs<1 || numbufs>BP_MAXN
       || nbpools >= NBPOOLS) {
    restore(mask);
    return (bpid32)SYSERR;
   }
   /* Round request to a multiple of 4 bytes */

   bufsiz = ( (bufsiz + 3) & (~3) );

   buf = (char *)getmem( numbufs * (bufsiz+sizeof(bpid32)) );
   if ((int32)buf == SYSERR) {
    restore(mask);
    return (bpid32)SYSERR;
   }
   poolid = nbpools++;
   bpptr = &buftab[poolid];
   bpptr->bpnext = (struct bpentry *)buf;
   bpptr->bpsize = bufsiz;
```

13

```
if ( (bpptr->bpsem = semcreate(numbufs)) == SYSERR) {
    nbpools--;
    restore(mask);
    return (bpid32)SYSERR;
   }
   bufsiz+=sizeof(bpid32);
   for (numbufs-- ; numbufs>0 ; numbufs-- ) {
    bpptr = (struct bpentry *)buf;
    buf += bufsiz;
    bpptr->bpnext = (struct bpentry *)buf;
   }
   bpptr = (struct bpentry *)buf;
   bpptr->bpnext = (struct bpentry *)NULL;
   restore(mask);
   return poolid;
}
```

14

# Initializing the Buffer Pool Table

```
/* bufinit.c - bufinit */

#include <xinu.h>

struct  bpentry buftab[NBPOOLS];      /* buffer pool table      */
bpid32  nbpools;

/*------------------------------------------------------------------------
 *  bufinit  --  initialize the buffer pool data structure
 *------------------------------------------------------------------------
 */
status  bufinit(void)
{
   nbpools = 0;
   return OK;
}
```

15

# end

16