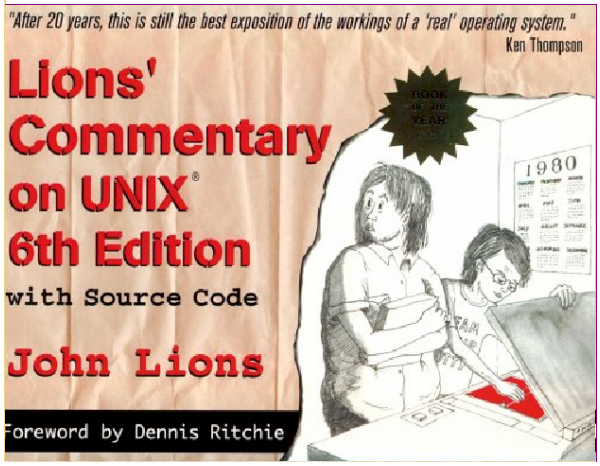
Chapter 5
"Two Files"

malloc.c and *prf.c*



Alexander Aved
Hao Cheng
3/28/2005

Overview

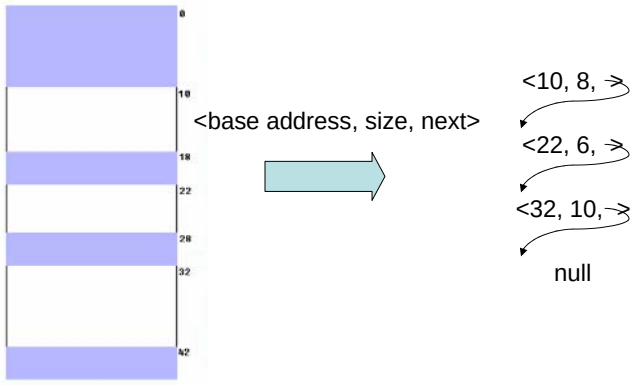
"Two Files"
- – malloc.c – Hao Cheng
  - malloc
  - mfree
- – prf.c – Alex Aved
  - Stack organization
  - putchar
  - Transmitter registers
  - printn
  - printf
  - prdev, deverror, panic

malloc.c

- malloc.c is used for memory management.
- Data structure:
  - struct map (2515)
    List of available resources (memory, disk swap area)
- Two procedures
  - malloc(2528)
    resources allocation
  - mfree(2556)
    resources release

Free memory - List

- Disjoint free memory regions



Kinds of memory resources

- main memory (64 bytes)
- disk swap area (512 bytes)
- separate lists, common interface

```
2515 struct map
2516 {
2517    char *m_size;
2518    char *m_addr;
2519 };
0141 #define CMAPSIZ 100     /* size of core allocation area */
0142 #define SMAPSIZ 100     /* size of swap allocation area */
0203 int  coremap[CMAPSIZ];     /* space for core allocation */
0204 int  swapmap[SMAPSIZ];     /* space for swap allocation */

2528 malloc(mp, size)
2529 struct map *mp;
2556 mfree(mp, size, aa)
2557 struct map *mp;
```

map structure

global list variables

function specification

map structure

- array represents list
  - first entry with the size of zero

```
2515 struct map
2516 {
2517    char *m_size;
2518    char *m_addr;
2519 };
```

no pointer

16-bit Address -> sizeof(map) = 4bytes

- cast from int to struct map

```
0141 #define CMAPSIZ 100     /* size of core allocation area */
0142 #define SMAPSIZ 100     /* size of swap allocation area */
0203 int  coremap[CMAPSIZ];     /* space for core allocation */
0204 int  swapmap[SMAPSIZ];     /* space for swap allocation */
```

int: 1 word -> sizeof(coremap) = 200bytes

Only 50 list entries

int, not map

memory management rules

- Memory Allocation
  - scan the list to find a suitable slot
  - First fit, Best fit and Worst fit
  - size of slot vs size of request
    - = delete the node
    - > reduce the size of the node
- Memory Recycle
  - create a represented node
  - merge if necessary

malloc

sequential scan, bp -> first fit region

update the linked list structure

First fit region = request

```
2528 malloc(mp, size)
2529 struct map *mp;
2530 {
2531     register int a;
2532     register struct map *bp;
2533
2534     for (bp = mp; bp->m_size; bp++) {
2535             if (bp->m_size >= size) {
2536                     a = bp->m_addr;
2537                     bp->m_addr =+ size;
2538                     if ((bp->m_size =- size) == 0)
2539                         do {
2540                             bp++;
2541                             (bp-1)->m_addr = bp->m_addr;
2542                         } while((bp-1)->m_size = bp->m_size);
2543                     return(a);
2544             }
2545     }
2546     return(0);
2547 }
```

mfree

```
2556 mfree(mp, size, aa)
2557 struct map *mp;
2558 {
2559     register struct map *bp;
2560     register int t;
2561     register int a;
2562
2563     a = aa;
2564     for (bp = mp; bp->m_addr<=a && bp->m_size!=0; bp++);
2565     if (bp>mp && (bp-1)->m_addr+(bp-1)->m_size == a) {
2566             (bp-1)->m_size =+ size;
2567             if (a+size == bp->m_addr) {
2568                     (bp-1)->m_size =+ bp->m_size;
2569                     while (bp->m_size) {
2570                             bp++;
2571                             (bp-1)->m_addr = bp->m_addr;
2572                             (bp-1)->m_size = bp->m_size;
2573                     }
2574             }
2575     } else {
2576             if (a+size == bp->m_addr && bp->m_size) {
2577                     bp->m_addr =- size;
2578                     bp->m_size =+ size;
2579             } else if(size) do {
2580                     t = bp->m_addr;
2581                     bp->m_addr = a;
2582                     a = t;
2583                     t = bp->m_size;
2584                     bp->m_size = size;
2585                     bp++;
2586             } while (size = t);
2587     }
2588 }
```

sequential scan, 1st region above returned region

bp not 1st list item, merge(preceding, returned)

merge(returned, following/bp)

opposite situation

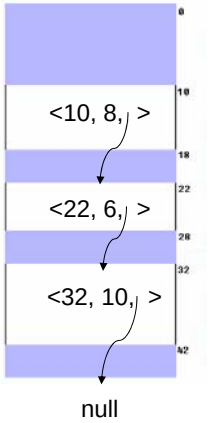
bp valid
merge(returned, bp)

no any merge
insert new entry

Example Scenario

- Initial list of free core memory resources

Entry	Size	Address
0	8	10
1	6	22
2	10	32
3	0	??
4	??	??



- 1st request: malloc(coremap, 6)

malloc(coremap, 6)

- 1st request:

coremap

bp->m_size > size

Entry	Size	Address
0	8	10
1	6	22
2	10	32
3	0	??
4	??	??

```
2528 malloc(mp, size)
2529 struct map *mp;
2530 {
2531     register int a;
2532     register struct map *bp;
2533
2534     for (bp = mp; bp->m_size; bp++) {
2535             if (bp->m_size >= size) {
2536                     a = bp->m_addr;
2537                     bp->m_addr =+ size;
2538                     if ((bp->m_size =- size) == 0)
2539                         do {
2540                             bp++;
2541                             (bp-1)->m_addr = bp->m_addr;
2542                         } while((bp-1)->m_size = bp->m_size);
2543                     return(a);
2544             }
2545     }
2546     return(0);
2547 }
```

malloc(coremap, 6)

- 1st request:

Entry	Size	Address
0	8	10
1	6	22
2	10	32
3	0	??
4	??	??

a =10
bp->m_addr = 16
bp->m_size = 2
return(10)

```
2528 malloc(mp, size)
2529 struct map *mp;
2530 {
2531     register int a;
2532     register struct map *bp;
2533
2534     for (bp = mp; bp->m_size; bp++) {
2535             if (bp->m_size >= size) {
2536                     a = bp->m_addr;
2537                     bp->m_addr =+ size;
2538                     if ((bp->m_size =- size) == 0)
2539                         do {
2540                             bp++;
2541                             (bp-1)->m_addr = bp->m_addr;
2542                         } while((bp-1)->m_size = bp->m_size);
2543                     return(a);
2544             }
2545     }
2546     return(0);
2547 }
```

malloc(coremap, 6)

- 2nd request:

Entry	Size	Address
0	2	16
1	6	22
2	10	32
3	0	??
4	??	??

```
2528 malloc(mp, size)
2529 struct map *mp;
2530 {
2531     register int a;
2532     register struct map *bp;
2533
2534     for (bp = mp; bp->m_size; bp++) {
2535             if (bp->m_size >= size) {
2536                     a = bp->m_addr;
2537                     bp->m_addr =+ size;
2538                     if ((bp->m_size =- size) == 0)
2539                         do {
2540                             bp++;
2541                             (bp-1)->m_addr = bp->m_addr;
2542                         } while((bp-1)->m_size = bp->m_size);
2543                     return(a);
2544             }
2545     }
2546     return(0);
2547 }
```

malloc(coremap, 6)

- 2nd request:

Entry	Size	Address
0	2	16
1	6	22
2	10	32
3	0	??
4	??	??

```
2528 malloc(mp, size)
2529 struct map *mp;
2530 {
2531     register int a;
2532     register struct map *bp;
2533
2534     for (bp = mp; bp->m_size; bp++) {
2535             if (bp->m_size >= size) {
2536                     a = bp->m_addr;
2537                     bp->m_addr =+ size;
2538                     if ((bp->m_size =- size) == 0)
2539                         do {
2540                             bp++;
2541                             (bp-1)->m_addr = bp->m_addr;
2542                         } while((bp-1)->m_size = bp->m_size);
2543                     return(a);
2544             }
2545     }
2546     return(0);
2547 }
```

a = 22
bp->m_addr = 28
bp->m_size = 0

malloc(coremap, 6)

- 2nd request:

```
2528 malloc(mp, size)
2529 struct map *mp;
2530 {
2531     register int a;
2532     register struct map *bp;
2533
2534     for (bp = mp; bp->m_size; bp++) {
2535             if (bp->m_size >= size) {
2536                     a = bp->m_addr;
2537                     bp->m_addr =+ size;
2538                     if ((bp->m_size =- size) == 0)
2539                         do {
2540                             bp++;
2541                             (bp-1)->m_addr = bp->m_addr;
2542                         } while((bp-1)->m_size = bp->m_size);
2543                     return(a);
2544             }
2545     }
2546     return(0);
2547 }
```

return(22)

Entry	Size	Address
0	2	16
1	10	32
2	0	32
3	0	??
4	??	??

mfree(coremap, 4, 24)

- Release resources

```
2556 mfree(mp, size, aa)
2557 struct map *mp;
2558 {
2559     register struct map *bp;
2560     register int t;
2561     register int a;
2562
2563     a = aa;
2564     for (bp = mp; bp->m_addr<=a && bp->m_size!=0; bp++);
2565     if (bp>mp && (bp-1)->m_addr+(bp-1)->m_size == a) {
2566             (bp-1)->m_size =+ size;
2567             if (a+size == bp->m_addr) {
2568                     (bp-1)->m_size =+ bp->m_size;
2569                     while (bp->m_size) {
2570                             bp++;
2571                             (bp-1)->m_addr = bp->m_addr;
2572                             (bp-1)->m_size = bp->m_size;
2573                     }
2574             }
2575     } else {
2576             if (a+size == bp->m_addr && bp->m_size) {
2577                     bp->m_addr =- size;
2578                     bp->m_size =+ size;
2579             } else if(size) do {
2580                     t = bp->m_addr;
2581                     bp->m_addr = a;
2582                     a = t;
2583                     t = bp->m_size;
2584                     bp->m_size = size;
2585                     bp++;
2586             } while (size = t);
2587     }
2588 }
```

Entry	Size	Address
0	2	16
1	10	32
2	0	??
3	0	??
4	??	??

mfree(coremap, 4, 24)

```
2556 mfree(mp, size, aa)
2557 struct map *mp;
2558 {
2559     register struct map *bp;
2560     register int t;
2561     register int a;
2562
2563     a = aa;
2564     for (bp = mp; bp->m_addr<=a && bp->m_size!=0; bp++);
2565     if (bp>mp && (bp-1)->m_addr+(bp-1)->m_size == a) {
2566             (bp-1)->m_size =+ size;
2567             if (a+size == bp->m_addr) {
2568                     (bp-1)->m_size =+ bp->m_size;
2569                     while (bp->m_size) {
2570                             bp++;
2571                             (bp-1)->m_addr = bp->m_addr;
2572                             (bp-1)->m_size = bp->m_size;
2573                     }
2574             }
2575     } else {
2576             if (a+size == bp->m_addr && bp->m_size) {
2577                     bp->m_addr =- size;
2578                     bp->m_size =+ size;
2579             } else if(size) do {
2580                     t = bp->m_addr;
2581                     bp->m_addr = a;
2582                     a = t;
2583                     t = bp->m_size;
2584                     bp->m_size = size;
2585                     bp++;
2586             } while (size = t);
2587     }
2588 }
```

Entry	Size	Address
0	2	16
1	10	32
2	0	??
3	0	??
4	??	??

bp>mp, but (16 + 2) != 24

24 + 4 != 32

create and insert one entry before bp
a and t act as temporary variables

mfree(coremap, 6, 18)

- 2nd mfree

```
2556 mfree(mp, size, aa)
2557 struct map *mp;
2558 {
2559     register struct map *bp;
2560     register int t;
2561     register int a;
2562
2563     a = aa;
2564     for (bp = mp; bp->m_addr<=a && bp->m_size!=0; bp++);
2565     if (bp>mp && (bp-1)->m_addr+(bp-1)->m_size == a) {
2566             (bp-1)->m_size =+ size;
2567             if (a+size == bp->m_addr) {
2568                     (bp-1)->m_size =+ bp->m_size;
2569                     while (bp->m_size) {
2570                             bp++;
2571                             (bp-1)->m_addr = bp->m_addr;
2572                             (bp-1)->m_size = bp->m_size;
2573                     }
2574             }
2575     } else {
2576             if (a+size == bp->m_addr && bp->m_size) {
2577                     bp->m_addr =- size;
2578                     bp->m_size =+ size;
2579             } else if(size) do {
2580                     t = bp->m_addr;
2581                     bp->m_addr = a;
2582                     a = t;
2583                     t = bp->m_size;
2584                     bp->m_size = size;
2585                     bp++;
2586             } while (size = t);
2587     }
2588 }
```

Entry	Size	Address
0	2	16
1	4	24
2	10	32
3	0	??
4	??	??

mfree(coremap, 6, 18)

```
2556 mfree(mp, size, aa)
2557 struct map *mp;
2558 {
2559     register struct map *bp;
2560     register int t;
2561     register int a;
2562
2563     a = aa;
2564     for (bp = mp; bp->m_addr<=a && bp->m_size!=0; bp++);
2565     if (bp>mp && (bp-1)->m_addr+(bp-1)->m_size == a) {
2566             (bp-1)->m_size += size;
2567             if (a+size == bp->m_addr) {
2568                     (bp-1)->m_size += bp->m_size;
2569                     while (bp->m_size) {
2570                             bp++;
2571                             (bp-1)->m_addr = bp->m_addr;
2572                             (bp-1)->m_size = bp->m_size;
2573                     }
2574             }
2575     } else {
2576             if (a+size == bp->m_addr && bp->m_size) {
2577                     bp->m_addr -= size;
2578                     bp->m_size += size;
2579             } else if(size) do {
2580                     t = bp->m_addr;
2581                     bp->m_addr = a;
2582                     a = t;
2583                     t = bp->m_size;
2584                     bp->m_size = size;
2585                     bp++;
2586             } while (size = t);
2587     }
2588 }
```

Entry	Size	Address
0	12	16
1	10	32
2	0	??
3	0	??
4	??	??

bp>mp, and (16 + 2) = 18

(16 + 8) == 24

mfree(coremap, 6, 10)

- 3rd mfree

```
2556 mfree(mp, size, aa)
2557 struct map *mp;
2558 {
2559     register struct map *bp;
2560     register int t;
2561     register int a;
2562
2563     a = aa;
2564     for (bp = mp; bp->m_addr<=a && bp->m_size!=0; bp++);
2565     if (bp>mp && (bp-1)->m_addr+(bp-1)->m_size == a) {
2566             (bp-1)->m_size += size;
2567             if (a+size == bp->m_addr) {
2568                     (bp-1)->m_size += bp->m_size;
2569                     while (bp->m_size) {
2570                             bp++;
2571                             (bp-1)->m_addr = bp->m_addr;
2572                             (bp-1)->m_size = bp->m_size;
2573                     }
2574             }
2575     } else {
2576             if (a+size == bp->m_addr && bp->m_size) {
2577                     bp->m_addr -= size;
2578                     bp->m_size += size;
2579             } else if(size) do {
2580                     t = bp->m_addr;
2581                     bp->m_addr = a;
2582                     a = t;
2583                     t = bp->m_size;
2584                     bp->m_size = size;
2585                     bp++;
2586             } while (size = t);
2587     }
2588 }
```

Entry	Size	Address
0	12	16
1	10	32
2	0	??
3	0	??
4	??	??

mfree(coremap, 6, 10)

```
2556 mfree(mp, size, aa)
2557 struct map *mp;
2558 {
2559     register struct map *bp;
2560     register int t;
2561     register int a;
2562
2563     a = aa;
2564     for (bp = mp; bp->m_addr<=a && bp->m_size!=0; bp++);
2565     if (bp>mp && (bp-1)->m_addr+(bp-1)->m_size == a) {
2566             (bp-1)->m_size += size;
2567             if (a+size == bp->m_addr) {
2568                     (bp-1)->m_size += bp->m_size;
2569                     while (bp->m_size) {
2570                             bp++;
2571                             (bp-1)->m_addr = bp->m_addr;
2572                             (bp-1)->m_size = bp->m_size;
2573                     }
2574             }
2575     } else {
2576             if (a+size == bp->m_addr && bp->m_size) {
2577                     bp->m_addr -= size;
2578                     bp->m_size += size;
2579             } else if(size) do {
2580                     t = bp->m_addr;
2581                     bp->m_addr = a;
2582                     a = t;
2583                     t = bp->m_size;
2584                     bp->m_size = size;
2585                     bp++;
2586             } while (size = t);
2587     }
2588 }
```

bp<mp

(10 + 6) = 16

Entry	Size	Address
0	~~12~~	~~16~~
1	10	32
2	0	??
3	??	??
4	??	??

Resources List Initialization

- in main.c

```
1560    i = *ka6 + USIZE;
1561    UISD->r[0] = 077406;
1562    for(;;) {
1563            UISA->r[0] = i;
1564            if(fuibyte(0) < 0)
1565                    break;
1566            clearseg(i);
1567            maxmem++;
1568            mfree(coremap, 1, i);
1569            i++;
1570    }
```

Initially i is 1st block of core memory

One entry - coremap
address = first block of core memory
size = the whole core memory.

```
1580    printf("Electric Company, Inc.\n");
1581
1582    maxmem = min(maxmem, MAXMEM);
1583    mfree(swapmap, nswap, swplo);
1584
1585    /*
1586     * set up system process
1587     */
1588
4697 int swplo 4000; /* cannot be zero */
4698 int nswap 872;
```

One entry - swapmap

Usage Example

- Memory allocation.

```
1893    n = rip->p_size;
1894    a1 = rip->p_addr;
1895    rpp->p_size = n;
1896    a2 = malloc(coremap, n);
1897    /*
1898     * If there is not enough core for the
1899     * new process, swap put the current process to
1900     * generate the copy.
1901     */
1902    if(a2 == NULL) {
1903            rip->p_stat = SIDL;
1904            rpp->p_addr = a1;
1905            savu(u.u_ssav);
1906            xswap(rpp, 0, 0);
1907            rpp->p_flag =| SSWAP;
1908            rip->p_stat = SRUN;
1909    } else {
1910    /*
1911     * There is core, so just copy.
1912     */
1913            rpp->p_addr = a2;
1914            while(n--)
1915                    copyseg(a1++, a2++);
1916    }
```

```
3234    a = malloc(swapmap, 1);
3235    if(a == NULL)
3236            panic("out of swap");
```

When fail

Unit of resources

- block – manipulation unit
  - example: copyseg (0695)

```
0694 /* ------------------------          */
0695 .globl     _copyseg
0696 _copyseg:
0697    mov     PS,-(sp)
0698    mov     UISA0,-(sp)
0699    mov     UISA1,-(sp)
0700    mov     $30340,PS
0701    mov     10(sp),UISA0
0702    mov     12(sp),UISA1
0703    mov     UISD0,-(sp)
0704    mov     UISD1,-(sp)
0705    mov     $6,UISD0
0706    mov     $6,UISD1
0707    mov     r2,-(sp)
0708    clr     r0
0709    mov     $8192.,r1
0710    mov     $32.,r2
0711 1:
0712    mfpi    (r0)+
0713    mtpi    (r1)+
0714    sob     r2,1b
0715    mov     (sp)+,r2
0716    mov     (sp)+,UISD1
0717    mov     (sp)+,UISD0
0718    mov     (sp)+,UISA1
0719    mov     (sp)+,UISA0
0720    mov     (sp)+,PS
0721    rts     pc
0722
```

```
sob:
        r2 = r2 – 1;
        if (r2 != 0) goto 1b.
mfpi:
        push(r0)
mtpi:
        pop(r1)
```

```
r2 = 32;
while (r2-- > 0){
        mem[r0++] = mem[r1++];
}
```

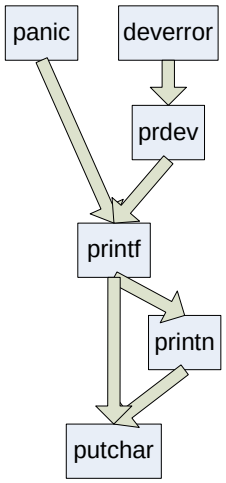
prf.c

- prf.c contains ‘panic’ and other procedures which provide a simple mechanism for displaying initialization messages and error messages to the operator.

•printf (2340)	•panic (2416)
•printn (2369)	•prdev (2433)
•putchar (2386)	•deverror (2447)

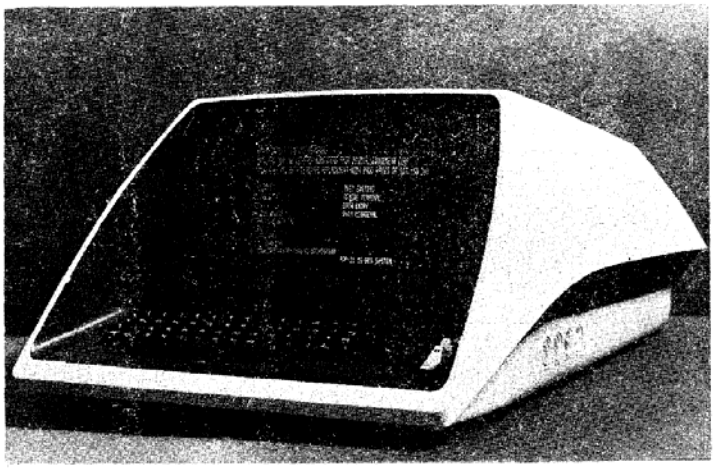
Note: printf & putchar similar to but not same as the versions invoked by C programs running in user mode.

Calling Relationship Between Procedures



PDP11/45 System Console

Alphanumeric VT05 Terminal



VT05 ALPHANUMERIC TERMINAL

Device Registers

```
0162 /* Certain processor registers */
0163
0164 #define PS 0177776
0165 #define KL 0177560
0166 #define SW 0177570
0167
0168 /* ------------------------       */
0169
0170 /* structures to access integers : */
0171
0172
0173     /* single integer */
0174
0175 struct {    int   integ;   };
0176
0177
0178    /* in bytes    */
0179
0180 struct {   char lobyte;   char hibyte;
0181
0182
0183    /* as a sequence */
0184
0185 struct {   int   r[];      };
0186
0187
0188 /* ------------------------       */
0189
```

Device interfaces

KL11 serial line controller

Console switch register

"Dummy" struct, used by putchar() to test if register pointed to by SW is 0.

KL11 IO Registers

```
2304 #include "../param.h"
2305 #include "../seg.h"
2306 #include "../buf.h"
2307 #include "../conf.h"
2308
2309 /*
2310  * Address and structure of the
2311  * KL-11 console device registers.
2312  */
2313 struct
2314 {
2315     int     rsr;
2316     int     rbr;
2317     int     xsr;
2318     int     xbr;
2319 };
2320 /* ------------------------       */
2321
2322 /*
2323  * In case console is off,
2324  * panicstr contains argument to last
2325  * call to panic.
2326  */
2327
2328 char        *panicstr;
2329
2330 /*
```

Receiver Status Register

Receiver Buffer Register

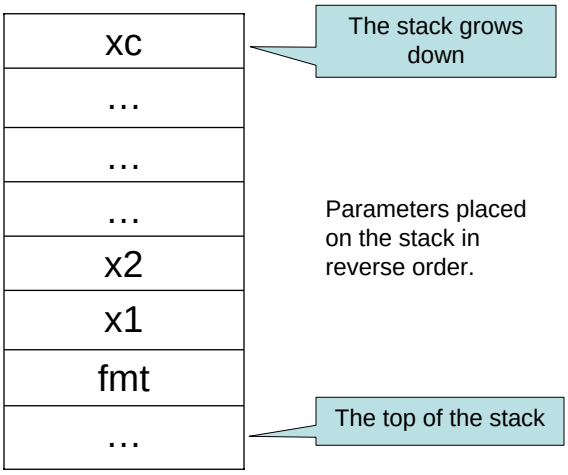
Transmitter Status Register

Transmitter Buffer Register

Can be read or written by any PDP11 instruction that refers to their address.

The Stack

printf(fmt,x1,x2,x3,x4,x5,x6,x7,x8,x9,xa,xb,xc)



putchar(c)

- putchar() transmits the character passed in its parameter to the system console.

putchar(c)

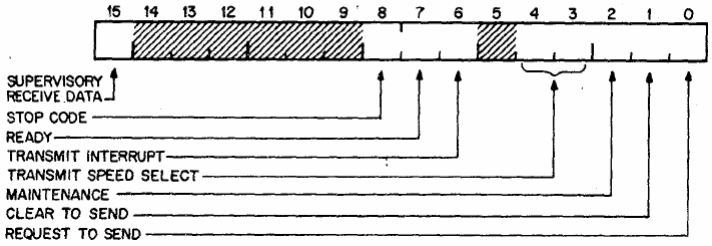
```
2386 putchar(c)
2387 {
2388     register rc, s;
2389
2390     rc = c;
2391     if(SW->integ == 0)
2392             return;
2393     while((KL->xsr&0200) ==0)
2394             ;
2395     if(rc == 0)
2396             return;
2397     s = KL->xsr;
2398     KL->xsr = 0;
2399     KL->xbr = rc;
```

```
2400     if(rc == '\n') {
2401             putchar('\r');
2402             putchar(0177);
2403             putchar(0177);
2404     }
2405     putchar(0);
2406     KL->xsr = s;
2407 }
2408 /* ------------------------
```

ASCII 'DEL', 0177, is Octal for 127_{10}

Transmitter Status Register

Transmitter Status Register (TSCR)

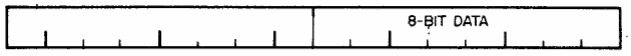


7	Ready	Indicates transmitter ready to output data. Cleared by loading transmitter buffer, set by having transmitter buffer zero'd. Read only.
6	Transmit Interrupt	Enables the transmitter interrupt facility. Read and Write.

NOTE: in "C" code, this register is referred to as "xsr", in Peripherals Handbook, referred to as "TSCR".

Transmitter Buffer Register

Transmitter Buffer Register (TBUF)



8.2.4 Specifications

Operating mode:	Full- or half-duplex selected under software control.
Data Rates:	50, 75, 110, 134.5, 150, 300, 600, 1200, 1800 Baud or one user specified Baud rate between 600 and 10,000. Four speeds are available to the user under program control. Transmitting and receiving rates are independent. See DC11 models for specific combinations available.
Data Format:	One start bit. Character size is variable under program control to 5, 6, 7, or 8 data bits. Stop code is programmable to one or two bits.
Order of Bit Transmission:	Low order bit first.
Parity:	Computed on incoming data.

NOTE: referred to as “xbr” in “C” code.

printn(n, b)

- printn() prints to the console an unsigned integer *n*, in base *b*.
- *That is, expresses n as a set of digit characters according to the radix b.*
- E.g., printn(10, 8) results in the following calls to putchar():
  - putchar(49);
  - putchar(50);

  Which prints to the console the characters “12”

printn(n, b)

```
2366 /*
2367  * Print an unsigned integer in base b.
2368  */
2369 printn(n, b)
2370 {
2371     register a;
2372
2373     if(a = ldiv(n, b))
2374             printn(a, b);
2375     putchar(lrem(n, b) + '0');
2376 }
2377 /* ------------------------        */
```

Integer divide

Note that this is the ASCII character '0', 48 base 10

Remainder of n / b
E.g., 1 / 8 = 0 rem 1

printf(fmt,x1,…,x9,xa,xb,xc)

- An **unbuffered** way for the operating system to send a message to the system console.
- Used during initialization & to report hardware errors or system problems.
- Runs in kernel mode.
- Accepts the following % flags:
  - l (ell), d, o and s

printf(fmt,x1,…,x9,xa,xb,xc)

```
2340 printf(fmt,x1,x2,x3,x4,x5,x6,x7,x8,x9,xa,xb,xc)
2341 char fmt[];
2342 {
2343     register char *s;
2344     register *adx, c;
2345
2346     adx = &x1;
2347 loop:
2348     while((c = *fmt++) != '%') {
2349             if(c == '\0')
2350                     return
2351             putchar(c);
2352     }
2353     c = *fmt++;
2354     if(c == 'd'  || c == 'l' || c == 'o')
2355             printn(*adx, c=='o'? 8: 10);
2356   if(c == 's') {
2357             s = *adx;
2358             while(c = *s++)
2359                     putchar(c);
2360     }
2361     adx++;
2362     goto loop;
2363 }
2364 /* ------------------------        */
```

'x1' points to a location on the stack

While not a '%', send the format string to the console.

Display a number. E.g., printf("ten: %d", 10)

Display a string. E.g., printf("Today: %s", "Monday")

Increment 'adx' to point to the next position on the stack

prdev(str, dev)

- The prdev() procedure provides a warning message when errors are occurring in i/o operations.
- E.g., the message will look something like
  - "*message* on dev *12*/*34*"

prdev(str, dev)

```
2427 /*
2428  * prdev prints a warning message of the
2429  * form "mesg on dev x/y".
2430  * x and y are the major and minor parts of
2431  * the device argument.
2432  */
2433 prdev(str, dev)
2434 {
2435
2436  printf("%s on dev %l/%l\n", str, dev.d_major, dev.d_minor);
2437 }
```

```
Sep  1 09:28 1988  unix/conf.h Page 1


4600 /* Used to dissect integer device code
4601  * into major (driver designation) and
4602  * minor (driver parameter) parts.
4603  */
4604 struct     {
4605            char    d_minor;
4606            char    d_major;
4607 };
```

The ‘d_major’ number is an index into a system table to select a device driver. ‘d_minor’ is passed as a parameter to specify a subdevice attached to a controller.

deverror(bp, o1, o2)

- This procedure provides a warning message when errors are occurring in i/o operations.
- deverror() prints a diagnostic message from a device driver.
  - Parameters:
    - bp – device
    - o1 – a block number
    - o2 – an octal word, e.g., an error status

deverror(bp, o1, o2)

```
2447 deverror(bp, o1, o2)
2448 int *bp;
2449 {
2450     register *rbp;
2451
2452     rbp = bp;
2453     prdev("err", rbp->b_dev);
2454     printf("bn%l er%o %o\n", rbp->b_blkno, o1, o2);
2455 }
2456 /* -------------------------       */
```

E.g., prints "*message* on dev *12*/*34*"

E.g., prints "bn*1234* er*567012 3456*"

The buffer header

```
4515  * A buffer header contains all the information required
4516  * to perform I/O.
4517  * Most of the routines which manipulate these things
4518  * are in bio.c.
4519  */
4520 struct buf
4521 {
4522     int     b_flags;        /* see defines below */
4523     struct  buf *b_forw;    /* headed by devtab of b_dev */
4524     struct  buf *b_back;    /*  "  */
4525     struct  buf *av_forw;   /* position on free list, */
4526     struct  buf *av_back;   /*     if not BUSY*/
4527     int     b_dev;          /* major+minor device name */
4528     int     b_wcount;       /* transfer count (usu. words) */
4529     char    *b_addr;        /* low order core address */
4530     char    *b_xmem;        /* high order core address */
4531     char    *b_blkno;       /* block # on device */
4532     char    b_error;        /* returned after I/O */
4533     char    *b_resid;       /* words not transferred after
4534                                        error */
4535 } buf[NBUF];
4536 /* -------------------------       */
```

panic(s)

- panic() is called from various locations within the operating system when circumstances exist such that continued operation of the system seems undesirable.
  - E.g., certain file system /device driver/etc. problems.
  - Go to http://sunsolve.sun.com/ and search with keyword 'panic' for Solaris examples.

panic(s)

```
2410 /*
2411  * Panic is called on unresolvable
2412  * fatal errors.
2413  * It syncs, prints "panic: mesg" and
2414  * then loops.
2415  */
2416 panic(s)
2417 char *s;
2418 {
2419     panicstr = s;
2420     update();
2421     printf("panic: %s\n", s);
2422     for(;;)
2423             idle();
2424 }
2425 /* ----------------------------- */
```

If the console is off, panicstr contains the argument to the last call of 'panic()'

update() is the internal name for the sync() system call.

update() *(1)* goes through the mount table & updates recently modified 'super blocks', *(2)* writes out updated inodes & *(3)* calls bflush() which forces to disk any "delayed write" blocks.

idle() halts the processor but allows already underway i/o operations to complete.

Summary – "Two Files"

- malloc.c
  - Routines to manage memory resources
    - malloc
    - mfree
- prf.c
  - Provides simple mechanism for displaying initialization and error messages to the operator
    - printf
    - printn
    - putchar
    - panic
    - prdev
    - deverror

References

- http://tibbitts.freeshell.org/photo/unix/pdp11-002.html
  - For console image
- http://www.ba-stuttgart.de/~helbig/os/script/OS.pdf
  - Unix V6 OS information
- http://www.bitsavers.org/pdf/Whatsnew.txt
  - Additional PDP manuals
- http://wolfram.schneider.org/bsd/7thEdManVol2/
  - UNIX V7 manuals
- J Lions, A Commentary on the Sixth Edition UNIX Operating System

Questions?