XINU System Calls

status ascdate(uint32 now, char *s)

Prepare in s a formatted ASCII string representing the time specified by now, which is the number of seconds past January 1, 1970 UTC. [This should not be a system call; it should be a library function. But the source code for this is in the system directory, so it is listed here.] The result in s will look similar to “Sep  28 18:17:41 CDT 2016” and require at least 31 bytes (including the null terminator).

status bufinit(void)

Initialize the buffer pool data structure. This function is to be invoked only once during system initialization, and sets the number of buffer pools to zero. Always returns OK.

pr16 chprio(pid32 pid, pr16 newprio)

Change the scheduling priority of a process. The pid argument identifies the process whose priority is to be changed, and newprio identifies the new priority.

Returns SYSERR if the pid argument is bad (that is, it does not identify an existing process). Otherwise it returns the old priority that was associated with the process.

syscall close(did32 descrp)

Close a device.

Returns SYSERR if the device descriptor is bad. Otherwise it returns the value returned by the dvclose entry of the device driver for the device.

syscall control(did32 descrp, int32 func, int32 arg1, int32 arg2)

Control a device or a driver (e.g. set the driver mode). The func, arg1, and arg2 arguments are passed to the dvcntl entry of the device driver for the device.

Returns SYSERR if the device descriptor is bad. Otherwise it returns the value returned by the dvcntl entry of the device driver for the device.

syscall create(void *funcaddr, uint32 ssize, pr16 priority, char *name, uint32 nargs, ...)

Create a new process to start running a named function.

On success, the process ID of the new process is returned, and the new process is left in the suspended state.

syscall freebuf(char *bufaddr)

Free a buffer that was allocated from a pool by getbuf. The bufaddr argument must be a value that was previously returned by getbuf.

Returns OK on success, or SYSERR if the buffer pool identification attached to the buffer is incorrect.

syscall freemem(char *blkaddr, int32 nbytes)

Free a memory block, returning the block to the free list. The lowest address of the block of memory to be freed is provided in the blkaddr argument. The number of bytes in the block to
be freed is given by the `nbytes` argument, which is rounded up, if necessary, so it is a multiple of 8 bytes.

A `freemem` invocation will return `SYSERR` if the block size is 0, or if the block specified by the two arguments does not lie wholly within the heap space managed by the system or if the block overlaps any existing free memory block. Otherwise it will return `OK`.

`char *getbuf(bpid32 poolid)`

Get a buffer from a pre-established buffer pool. This function will block the caller until a buffer is available.

On success, the address of a buffer is returned. If the buffer pool ID in `poolid` is incorrect, `(char *)SYSERR` is returned.

`syscall getc(did32 descrp)`

Obtain one byte from a device by calling the `dvgetc` entry of the device driver for the device.

Returns `SYSERR` if the device descriptor is bad. Otherwise it returns the value returned by the `dvgetc` entry of the device driver for the device.

`char *getmem(uint32 nbytes)`

Allocate `nbytes` of heap storage. On success, the address of the lowest byte in the allocated region of storage is returned. On error, `(char *)SYSERR` is returned.

`pid32 getpid(void)`

Return the process ID of the currently executing process.

`syscall getprio(pid32 pid)`

Return the scheduling priority of a process. There is possibility of an error return.

`void hexdump(void *buffer, uint32 length, bool8 canon)`

Write a hexadecimal, and optionally an ASCII, display of a region of memory to the standard output device. The region of memory to be displayed starts at the address pointed to by the `buffer` argument, and contains `length` bytes. If the `canon` argument is `FALSE`, then only the hexadecimal display appears. Otherwise, both the hexadecimal and ASCII displays will appear.

`syscall init(did32 descrp)`

Initialize a device and its driver by calling the `dvinit` entry of the device driver for the device identified by `descrp`.

Returns `SYSERR` if the device descriptor is bad. Otherwise it returns the value returned by the `dvinit` entry of the device driver for the device.
syscall kill(pid32 pid)

Kill the process with the specified pid and remove it from the system. If pid does not identify an existing process in any state except the null/idle process, kill will return SYSERR. If the process being killed is the last process in the system except the null/idle process, the system is halted.

Otherwise a message containing pid is sent to the parent of process pid, and the close system call is invoked for the device descriptors associated with the standard input, standard output, and standard error devices.

Killing the last process in the system (that is, only the idle process remains) cause the system to be halted by calling xdone.

syscall kputc(byte c)

Wait until the console serial port is not busy and then write the character in c to it. If c is a newline character ('\n'), then a carriage return character ('\r') is also written to the console.

Return SYSERR if there is no console device. Otherwise return OK.

kputc should normally only be used for output from the operating system kernel, or in other cases where the output must not be buffered (as it is with mechanisms like printf).

syscall kgetc(void)

Return SYSERR if there is no console device. Otherwise disable serial port interrupt recognition, wait (by looping) until a character is ready to be read from the console device, then read the character, restore interrupt recognition, and return the character just read.

kgetc should normally only be used when input is required during system initialization, or for debugging the kernel.

syscall kprintf(char *fmt, ...)

Format and print on the console device using polled output. The kputc function is used to write each character of the output. The fmt format argument is treated in the same manner as when the printf function is used. Always returns OK.

kprintf should normally only be used for output from the operating system kernel, or in other cases where the output must not be buffered (as it is with mechanisms like printf).

syscall netup(int32 k)

Get or set the system network status. If k is 0 or 1 the network status is set to 0 or 1 (disabled or enabled), and 0 or 1 is returned, respectively. If k is 2, the current network status is returned. Other values of k will cause SYSERR to be returned.

The network status, recorded in the global variable network_status, is used to enable or disable the use of the network by various applications and system services. For example, if the network is disabled, then the date and time related system calls will not attempt to contact a network time server.

syscall open(did32 descrp, char *name, char *mode)

Open a device.
syscall ptcreate(int32 count)
  Create a port that allows “count” outstanding messages.

syscall ptdelete(int32 portid, int32 (*disp)(int32))
  Delete a port, freeing waiting processes and messages.

syscall ptinit(int32 maxmsgs)
  Initialize all ports.

syscall ptreset(int32 portid, int32 (*disp)(int32))
  Reset a port, freeing waiting processes and messages and leaving the port ready for further use.

syscall ptsend(int32 portid, umsg32 msg)
  Send a message to a port by adding it to the queue.

syscall putc(did32 descrp, char ch)
  Send one character of data (byte) to a device.

syscall read(did32 descrp, char *buffer, uint32 count)
  Read one or more bytes from a device.

syscall seek(did32 descrp, uint32 pos)
  Position a random access device.

syscall semcount(sid32 semid)
  Return the count of a semaphore.

syscall semdelete(sid32 sem)
  Delete a semaphore by releasing its table entry.

syscall semreset(sid32 sem, int32 count)
  Reset a semaphore’s count and release waiting processes.

syscall send(pid32 pid, umsg32 msg)
  Pass a message to a process and start recipient if waiting.

syscall signal(sid32 sem)
  Signal a semaphore, releasing a process if one is waiting.

syscall signaln(sid32 sem, int32 count)
  Signal a semaphore n times, releasing n waiting processes.

syscall sleep(uint32 delay)
  Delay the calling process n seconds.

syscall sleepms(uint32 delay)
  Delay the calling process n milliseconds.

syscall stacktrace(pid32 pid)
  Print a stack backtrace for a process.
syscall suspend(pid32 pid)

Suspend a process, placing it in hibernation.

syscall wait(sid32 sem)

Cause current process to wait on a semaphore.

syscall write(did32 descrp, char *buffer, uint32 count)

Write one or more bytes to a device.

syscall yield(void)

Voluntarily relinquish the CPU (end a timeslice).
XINU Library Functions

int abs(int arg)
Return the absolute value of an integer.

int atoi(char *p)
Convert an ASCII string representation of a number to an integer. An acceptable ASCII string consists of any number of leading blanks and tabs, an optional sign (’+’ or ’-’) and any number of decimal digits. There is no check to see if the string has at least one decimal digit, and the conversion stops after the first non-digit character is encountered. A more aggressive conversion is performed by the strtol function.

long atol(char *p)
Convert an ASCII string representation of a number to a long integer. An acceptable ASCII string consists of any number of leading blanks and tabs, an optional sign (’+’ or ’-’) and any number of decimal digits. There is no check to see if the string has at least one decimal digit, and the conversion stops after the first non-digit character is encountered. A more aggressive conversion is performed by the strtol function.

void bzero(void *p, int len)
Set each of the len bytes starting at p to zero (that is, 0x00 not ’0’). The pointer p is not checked for validity.

int fgetc(int dev)
Get a character from device dev and return it. Return EOF on any error (that is, when getc on dev returns a negative result).

char *fgets(char *s, int n, int dev)
Get a newline-terminated string from device dev and store it starting at s. At most n-1 characters are read. The end of line termination character (either ’\n’ or ’\r’) is included in the string, if space permits. A null byte (’\0’) is stored after the last character in the buffer. The function returns s on success, and NULL on error, when n < 1, or when the end of file is encountered but no characters have been read.

int fprintf(int dev, char *fmt, ...)
Prepare a formatted string according to the format given by fmt and write it to device dev. Return 0 on success and -1 on error. The description of fmt is given in the description of the printf function.

int fputc(int c, int dev)
Write the character in c to device dev. Return c on success, and EOF on error.

int fputs(char *s, int dev)
Write the null-terminated string at s without its terminating null byte to device dev. Return the result of the putc call that writes the last character from s.
int fscanf(int dev, char *fmt, int args, ...)  
    Read from device dev and scan the input according to fmt. Store the results in the args. XXX THIS NEEDS MORE WORK. IN PARTICULAR, IS args THE NUMBER OF ARGUMENT POINTERS, OR THE FIRST ARGUMENT POINTER?

int getchar(void)  
    Read a character from the standard input device using fgetc. The result is the same as that returned by fgetc.

int isatty(int dev)  
    Returns 1 if device dev is a terminal (console) and 0 otherwise. If dev is not a valid device then SYSERR is returned.

int isdigit(int c)  
    Returns 1 if c is a digit ('0' .. '9') and 0 otherwise.

long labs(long arg)  
    Returns the absolute value of a long integer.

void longjmp(jmp_buf env, int val)  
    The storage associated with the env argument must have been previously established by a call to the setjmp function. When longjmp is called, it causes it to appear that the original setjmp function call returns again, but this time with the value val. In particular, the stack frame in effect when the setjmp function appears to return a second (or subsequent) time will be the same as that which was in effect when the original call was made to setjmp.

void *memchr(const void *s, int c, int n)  
    Scans the n bytes starting at s for the first occurrence of c, treating everything with the type unsigned char. Returns a pointer to the matching byte, or NULL if c was not found in the memory area.

int memcmp(const void *s1, const void *s2, int n)  
    Compare the n bytes starting at s1 and s2. If all pairs of the corresponding bytes from the two memory regions are the same, then memcmp returns 0. Otherwise it returns the signed difference between the first two bytes that are not the same.

void memcpy(void *s1, const void *s2, int n)  
    Copy the n bytes starting at s2 to s1.

void *memset(void *s, int c, int n)  
    Set the n bytes starting at s to c treated as an unsigned char. Returns s.

int printf(char *fmt, ...)  
    Prepare a formatted string according to the format given by fmt and write it to the standard output device. Return 0 on success and -1 on error.

int putchar(int c)  
    Write the character to the standard output device using fputc. The result is the same as that returned by fputc.
void qsort(void *base, unsigned int n, unsigned int size, int (*cmp)(const void *, const void *))

Sorts an array with n elements of size size. base points to the start of the array. The array is sorted into ascending order according to the comparison between elements performed by the function pointed to by cmp. This function, when passed pointers to two elements of the array, will return an integer less than, equal to, or greater than zero if the element pointed to by the first argument is respectively less than, equal to, or greater than the element pointed to by the second argument.

int rand(void)

Returns a pseudo-random integer in the range 0 to 32767. A linear congruential random number generator is used. If the initial seed is not specified with the srand function, then an initial seed of 1 will be used. This function is not reentrant or thread-safe.

int setjmp(jmp_buf env)

Save the information needed to preserve the current function call environment (corresponding to the current stack frame) of the calling process in the storage associated with the env argument and return 0.

int sprintf(char *s, char *fmt, ...)

Prepare a formatted string according to the format given by fmt and store it as a null-terminated string starting at s. Return 0 on success and -1 on error. XXX THE RETURN VALUE DOESN'T APPEAR TO MATCH THE CODE.

void srand(int seed)

Change the seed for the rand function to seed.

int sscanf(char *s, char *fmt, int args, ...)

Scan the null-terminated string at s according to fmt. Store the results in the args. XXX RELATE THIS TO FSCANF.

char *strcat(char *s1, const char *s2)

Concatenate the null-terminated string at s2 to the end of the null-terminated string at s1. Return s1.

char *strchr(const char *s, int c)

Return a pointer to the first occurrence of the character c in the null-terminated string at s. NULL is returned if no match is found.

int strcmp(const char *s1, const char *s2)

Compare the null-terminated strings at s1 and s2. Return an integer less, equal to, or greater than zero if s1 is respectively less than, equal to, or greater than s2.

char *strcpy(char *s1, const char *s2)

Copy the null-terminated string at s2 to the storage starting at s1, including the terminating null byte. Returns s1.
int strlen(const char *s)

    Return the length of the null-terminated string at s. This is the number of bytes from s up to but not including the null terminating byte.