Section 3: Library Functions

This section of the manual describes the functions available to programs from the standard Xinu library. C programmers will recognize some of the C library functions (esp. those that manipulate strings). Be careful: not all function arguments are like those in UNIX.

NAME

ascdate - print a date in ASCII including hours:mins:secs

SYNOPSIS

int ascdate(time, str)
long time;
char *str;

DESCRIPTION

Ascdate prints the date and time specified in *time* as an ASCII string which is copied to *str*. The string *str* must be long enough to hold the converted date and time.

SEE ALSO

gettime(2)

NAME

atoi, atol - extract and return an integer or long from a string

SYNOPSIS

int atoi(string)
char *string;

long atol(string)
char *string;

DESCRIPTION

Each function extracts the data type it returns from the string *string*. *Atol* requires that its function prototype be specified in the calling function.

SEE ALSO

sscanf(3)

NAME

blkcmp - lexical block comparison

SYNOPSIS

```
int blkcmp(s1, s2, len)
char *s1;
char *s2;
int len;
```

DESCRIPTION

Blkcmp compares up to *len* bytes of the blocks *s1* and *s2*. If the blocks are equal, the value 0 is returned. If the blocks differ, a negative integer is returned if a byte in block *s1* is less than the corresponding byte in block *s2*, otherwise a positive integer is returned.

SEE ALSO

blkequ(3)

NAME

blkcopy - copy a contiguous block of bytes

SYNOPSIS

int blkcopy(to, from, nbytes)
char *to;
char *from;
int nbytes;

DESCRIPTION

Blkcopy copies a block of *nbytes* contiguous bytes starting at location *from* into the area starting at location *to*. *Blkcopy* returns OK to the caller. *Blkcopy* will copy any byte value including the null character (zero).

SEE ALSO

blkequ(3), string(3)

BUGS

Blkcopy does not check for valid memory addresses.

NAME

blkequ - compare two contiguous blocks of memory for equality

SYNOPSIS

int blkequ(first, second, nbytes)
char *first;
char *second;
int nbytes;

DESCRIPTION

Blkequ compares two blocks of memory for equality. Each block contains exactly *nbytes* bytes. *Blkequ* returns FALSE if the two blocks differ, and TRUE if the blocks are equal. *Blkequ* compares all byte values including null (zero).

SEE ALSO

blkcmp(3), blkcopy(3), string(3)

NAME

ckmode - check a file mode string and convert to integer representation

SYNOPSIS

#include <file.h>

int ckmode(mode)
char *mode;

DESCRIPTION

Ckmode parses a null-terminated string, *mode*, containing characters that represent file modes, and produces an integer with mode bits set. The possible mode characters are:

r The file is to be opened for reading (i.e., input).

w The file is to be opened for writing (i.e., output).

n The file must be new. That is, it must not already exist.

o The file must be old. That is, it must already exist.

The file mode string, *mode*, can specify that the file is to be accessed for both reading and writing, but it cannot specify the mode to be both old and new. If neither reading nor writing is specified, *ckmode* assumes the file will be used for both. Similarly, if neither old nor new files are specified, *ckmode* assumes either is allowed.

Given a legal mode string, *ckmode* returns an integer with bits FLREAD, FLWRITE, FLOLD, and FLNEW set according to the argument, *mode*. *Ckmode* returns SYSERR if it finds illegal or duplicated characters in the argument string, or if the mode string specifies that the file must be both old and new.

SEE ALSO

access(2), open(2)

NAME

ctype - character type predicates and manipulation routines

SYNOPSIS

#include <ctype.h>

isalpha(c)
...
toascii(c)
tolower(c)
toupper(c)
char c;

DESCRIPTION

Routines beginning with *is* are predicates that classify the type of a character. Routines beginning with *to* convert characters. Each predicate returns TRUE if the condition is satisfied, and FALSE otherwise. In the current implementation, predicates are macros that use table lookup for efficiency.

isalnum	*c* is an alphanumeric character (i.e., a letter or digit)
isalpha	*c* is a lower- or upper-case letter
isascii	*c* is an ASCII character, code less than 0200
iscntrl	*c* has a value less than octal 040 or is a DEL (octal value 0177).
isdigit	*c* is a digit.
islower	*c* is a lower case letter.
isprint	*c* is a printable character with octal value 040 (blank) to 0176 (tilde).
isprshort	*c* is a printable short.
ispunct	*c* is a punctuation character (neither control nor alphanumeric).
isspace	*c* is a space, tab, carriage return, newline, or formfeed.
isupper	*c* is an upper case letter.
isxdigit	*c* is a hexadecimal digit (i.e., is 0-9 or a-f).
toascii	Converts *c* to an ascii by turning off high-order bits.
tolower	Converts argument *c* from upper to lower case.

toupper Converts argument *c* from lower to upper case.

Converts a

NAME

disable, enable, restore - change and restore processor interrupt status

SYNOPSIS

#include<kernel.h>
disable(ps);
int ps;

enable();

restore(ps);
int ps;

DESCRIPTION

These routines change the processor interrupt status mode. Normally, procedures use *disable* and *restore* to save the interrupt status, mask interrupts off, and then restore the saved status. *Enable* explicitly enables interrupts; **it is used only at system startup**.

NAME dot2ip - convert dotted decimal notation to an IP address

SYNOPSIS

int dot2ip(ip, pdot)
IPaddr ip;
char *pdot;

DESCRIPTION

Function *dot2ip* converts an Internet (IP) address from dotted decimal notation to its 32-bit integer form and stores it in argument *ip*. Argument *pdot* is a pointer to the null-terminated dotted decimal representation of the Internet address.

SEE ALSO

ip2dot(3), ip2name(2), name2ip(2)

NAME

fgetc, getchar - get character from a device

SYNOPSIS

```
#include <io.h>

int fgetc(dev)
int dev;

int getchar()
```

DESCRIPTION

These procedures are included for compatibility with UNIX. *Fgetc* returns the next character from the named input device.

Getchar() is identical to *getc(STDIN)*.

Note that *fgetc* is exactly equivalent to *getc*.

SEE ALSO

getc(2), putc(2), gets(3), scanf(3),

DIAGNOSTICS

These functions return SYSERR to indicate an illegal device or read error.

NAME

fputc, putchar - put character to a device

SYNOPSIS

```
#include <io.h>

int fputc(device, c)
int device;
char c;
putchar(c)
```

DESCRIPTION

These procedures are included for compatibility with UNIX. *Fputc* appends the character *c* to the named output device, and returns SYSERR if *device* is invalid; it is defined to be *putc(device, c)*.

Putchar(c) is defined to be *putc(STDOUT, c)*.

SEE ALSO

putc(2), puts(3), printf(3)

NAME

gets, fgets - get a string from a device

SYNOPSIS

```
#include <io.h>

char *gets(s)
char *s;

char *fgets(dev, s, n)
int  dev;
char *s;
int  n;
```

DESCRIPTION

Gets reads a string into *s* from the standard input device, CONSOLE. The string is terminated by a newline character, which is replaced in *s* by a null character. *Gets* returns its argument.

Fgets reads *n*-1 characters, or up to a newline character, whichever comes first, from device *dev* into the string *s*. The last character read into *s* is followed by a null character. *Fgets* returns its second argument.

SEE ALSO

getc(2), puts(2), scanf(3)

DIAGNOSTICS

Gets and *fgets* return SYSERR if an error results.

BUGS

Gets deletes a newline, *fgets* keeps it, all in the name of backward compatibility.

NAME

gpq - generic priority queue processing functions

SYNOPSIS

```
#include <q.h>

char *deq(q)
int q;

int enq(q, item, key)
int q;
char *item;
int key;

int freeq(q)
int q;

int newq(size, type)
int size;
int type;

char *seeq(q)
int q;
```

DESCRIPTION

The Xinu Kernel uses the queue structure QUEUE(3) to define and manipulate doubly-linked lists of processes. The networking extensions in Xinu 7.9 use Generic Priority Queues (GPQ) for queueing network packets. The *gpq* functions manipulate priority queues with mutual exclusion either with a semaphore, or by using *disable()/restore(),* depending on argument *type* when the queue is created. The supported operations are:

deq

Remove the first item from the list *q* and return it.

enq

Insert item *item* on the ordered list *q*, using integer *key* to choose a position for the item. Integer *key* represents the priority of an item. *Enq* returns the number of available queue element slots after the insertion, or SYSERR on overflow.

freeq

Delete the list *q*, returning all memory to the free list. If *q* is not empty, *freeq* returns SYSERR and does not free the list. Otherwise OK is returned.

newq

Allocate a new list that can hold up to *size* nodes, use *type* to determine whether mutual exclusion for the list is controlled with a semaphore (*type* is QF_WAIT) or by disabling interrupts (*type* is QF_NOWAIT), and return the list index. Queues which will be manipulated at interrupt time should specify QF_NOWAIT.

seeq

Search list *q* one item at a time, without removing the items; the list only remembers one search position at any instant. Multiple processes scanning a queue will see only some of the elements; no locking is done across *seeq* calls, and the queue may change from one call to the next because of *enq* or *deq* calls. Returns the items until all seen, then returns NULLPTR and resets *q_seen*.

A function *initq* is used **(only at system startup)** to initialize the GPQ mechanism.

SEE ALSO

queue(3)

NAME ip2dot - convert an IP address to dotted decimal notation

SYNOPSIS

```
#include <network.h>

char *ip2dot(pdot, ip)
char *pdot;
IPaddr ip;

int ip2str(cp, ip)
char *cp;
IPaddr ip;
```

DESCRIPTION

Function *ip2dot* converts an Internet (IP) address from its 32-bit integer form to dotted decimal notation and stores it in argument *pdot*. Argument *pdot* should be a pointer to a buffer at least 16 bytes long, the maximum length of a dotted decimal address and its terminating null character The function returns *pdot*.

Function *ip2str* performs the same conversion, but returns OK.

SEE ALSO

dot2ip(3), name2ip(2), ip2name(2)

NAME

netnum - compute the network portion of a given Internet (IP) address

SYNOPSIS

#include <network.h>
int netnum(netpart, address)
IPaddr netpart;
IPaddr address;

DESCRIPTION

Netnum extracts the network portion of the Internet address specified by argument *address*, and places the result in argument *netpart*. It operates by using the IP class of argument *address* to determine whether the network part of the address occupies 1, 2, or 3 bytes, and it zeros the remaining bytes. *Netnum* always returns OK.

SEE ALSO

getaddr(2), getnet(2)

BUGS

Netnum does not understand subnets or subnet masks.

NAME

netutil - Network utilities hs2net, net2hs, hl2net, net2hl, hl2vax, vax2hl

SYNOPSIS

```
#include <network.h>

short net2hs(s)
short hs2net(s)
long net2hl(l)
long hl2net(l)
long vax2hl(l)
long hl2vax(l)

short s;
long l;
```

DESCRIPTION

These routines map binary integer data between network standard byte order and local host byte order. In the description, the term *short* refers to a 2-octet (16-bit) binary value, whether two's complement signed or unsigned, and the term *long* refers to a 4-octet (32-bit) value.

net2hs

Converts a short item from network byte order to host byte order.

hs2net

Converts a short item from host byte order to network byte order.

net2hl

Converts a long item from network byte order to host byte order.

hl2net

Converts a long item from host byte order to network byte order.

vax2hl

Converts a long item from VAX byte order to host byte order (used in communication with a file server running on a VAX).

hl2vax

Converts a long from host byte order to VAX byte order (used in communication with a file server running on a VAX).

No conversion is needed for character strings because the local host order on most machines agrees with network standard byte order (i.e., the string extends upward in the memory address space).

NAME

pause - pause the processor

SYNOPSIS

pause();

DESCRIPTION

Pause stops the processor until an interrupt occurs, allowing it to continue at the instruction following the *pause* when the interrupt returns. *Pause* is used in the null process instead of an infinite loop to avoid taking bus bandwidth needlessly.

SEE ALSO

disable(3), wait(2), xdone(2)

NAME

printf, fprintf, sprintf - formatted output conversion

SYNOPSIS

printf(format [, arg] ...)
char *format;

fprintf(dev, format [, arg] ...)
int dev;
char *format;

sprintf(s, format [, arg] ...)
char *s, format;

DESCRIPTION

Printf writes formatted output on device STDOUT. *Fprintf* writes formatted output on the named output device. *Sprintf* places formatted 'output' in the string *s*, followed by the character '\0'.

Each of these functions converts, formats, and prints its arguments after the format under control of the *format* argument. The format argument is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of the next successive *arg*.

Each conversion specification is introduced by the character **%**. Following the %, there may be, in the following order,

- an optional minus sign '-' which specifies left adjustment of the converted value in the indicated field;

- an optional digit string specifying a field width; if the converted value has fewer characters than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width; if the field width begins with a zero, zero padding will be done instead of blank padding;

- an optional period '**.**' which serves to separate the field width from the next digit string;

- an optional digit string specifying a precision which specifies the maximum number of characters to be printed from a string;

- the character **l** specifying that a following **d**, **o**, **x**, or **u** corresponds to a long integer *arg*. (A capitalized conversion code accomplishes the same thing.)

- a character which indicates the type of conversion to be applied.

A field width or precision may be '*' instead of a digit string. In this case an integer *arg* supplies the field width or precision.

The conversion characters and their meanings are:

dox	The integer arg is converted to decimal, octal, or hexadecimal notation respectively.
c	The character arg is printed. Null characters are ignored.
s	*Arg* is taken to be a string (character pointer) and characters from the string are printed until a null character or until the number of characters indicated by the precision specification is reached; however if the precision is 0 or missing all characters up to a null are printed.
u	The unsigned integer arg is converted to decimal and printed (the result will be in the range 0 through 65535 on the PC for normal integers and 0 through 4294967295 for long integers).
%	Print a '%'; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width. Characters generated by *printf* are printed by PUTC(2).

Examples

To print a date and time in the form 'Sunday, July 3, 10:02', where *weekday* and *month* are pointers to null-terminated strings:

```
printf("%s, %s %d, %02d:%02d", weekday, month, day, hour, min);
```

SEE ALSO

putc(2), scanf(3)

BUGS

Very wide fields (>128 characters) fail.

NAME

puts, fputs - write a string to a device

SYNOPSIS

puts(s)
char *s;

fputs(dev, s)
int dev;
char *s;

DESCRIPTION

Puts writes the null-terminated string *s* on the output device STDOUT and appends a newline character.

Fputs writes the null-terminated string *s* on device *dev*.

Neither routine writes the terminal null character. They return SYSERR if *dev* is invalid.

SEE ALSO

gets(3), putc(3), printf(3), read(2), write(2)

BUGS

Puts appends a newline, *fputs* does not; there is no good reason for this.

NAME

qsort - quicker sort

SYNOPSIS

qsort(base, nel, width, compar)
char *base;
int (*compar)();

DESCRIPTION

Qsort is an implementation of the quicker-sort algorithm. The first argument is a pointer to the base of the data; the second is the number of elements; the third is the width of an element in bytes; the last is the name of the comparison routine to be called with two arguments which are pointers to the elements being compared. The routine must return an integer less than, equal to, or greater than 0 according as the first argument is to be considered less than, equal to, or greater than the second.

NAME

queue - q-structure predicates and list manipulation procedures

SYNOPSIS

```
#include <q.h>

int enqueue(proc, tail)
int dequeue(proc)
int firstid(head)
int firstkey(head)
int getfirst(head)
int getlast(tail)
int insert(proc, head, key)
int insertd(proc, head, key)
int isempty(head)
int lastkey(tail)
int nonempty(head)

int head, tail;
int proc;
int key;
```

DESCRIPTION

The *q* structure holds doubly-linked lists of processes, including lists of processes that are ready, sleeping, and waiting on a semaphore. These routines manipulate lists in the *q* structure as follows.

enqueue

Add a process to a FIFO list given the process id in argument *proc* and the q index of the tail of the list in argument *tail*. *Enqueue* returns argument *proc* to its caller.

dequeue

Remove a process from a list given the process id. The list on which the process is found need not be specified because it can be determined from the *q* structure. *Dequeue* will remove a process from both FIFO and ordered lists. It returns its argument to the caller.

firstid

Return the process id of the first process on a list given the *q* index of the list head in argument *head*.

firstkey

Return the integer key associated with the first entry on a list given the *q* index of the list in argument *head*.

getfirst

Remove the first process from a list and return its process id given the *q* index of the head of the list in argument *head*. *Getfirst* returns EMPTY if the list is empty, and a process id otherwise.

getlast

Remove the last process on a list and return its process id given the *q* index of the tail of the list in argument *tail*. *Getlast* returns EMPTY if the list is empty, and a process id otherwise.

insert

Insert a process into an ordered list given the process id in argument *proc*, the *q* index of the head of the list in argument *head*, and an integer key for the process in argument *key*. Ordered lists are always ordered by increasing key values. *Insert* returns OK.

insertd

Insert a process in a delta list given the process id in argument *proc*, the *q* index of the head of the list in argument *head*, and an integer key in argument *key*. *Insertd* returns OK.

isempty

Return TRUE if there are no processes on a list, FALSE otherwise, given the *q* index of the head of the list in argument *head*.

lastkey

Return the key of the last process in a list given the *q* index of the tail of the list in argument *tail*.

nonempty

Return TRUE if there is at least one process on a list, FALSE otherwise, given the *q* index of the head of the list in argument *head*.

SEE ALSO

gpq(3)

BUGS

Most of these routines do not check for valid arguments or valid lists. Also, they assume interrupts are disabled when called, and will corrupt the list structure if the caller fails to disable interrupts.

NAME

rand, srand - random number generator

SYNOPSIS

srand(seed)
int seed;

rand()

DESCRIPTION

Rand uses a multiplicative congruential random number generator with period 2^{32} to return successive pseudo-random numbers in the range from 0 to 2^{31}-1.

The generator is reinitialized by calling *srand* with 1 as argument. It can be set to a random starting point by calling *srand* with whatever you like as argument.

BUGS

Rand does not provide mutual exclusion among calling processes. Thus, there is a small chance that two concurrent processes will receive the same value.

NAME

scanf, fscanf, sscanf - formatted input conversion

SYNOPSIS

```
scanf(format [ , pointer ] . . .  )
char *format;

fscanf(dev, format [ , pointer ] . . .  )
int dev;
char *format;

sscanf(s, format [ , pointer ] . . .  )
char *s, *format;
```

DESCRIPTION

Scanf reads from the standard input device STDIN. *Fscanf* reads from the named input device. *Sscanf* reads from the character string *s*. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects as arguments a control string format, described below, and a set of pointer arguments indicating where the converted input should be stored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

- Blanks, tabs or newlines, which match optional white space in the input.

- An ordinary character (not %) which must match the next character of the input stream.

- Conversion specifications, consisting of the character **%**, an optional assignment suppressing character *****, an optional numerical maximum field width, and a conversion character.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by *. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. The following conversion characters are legal:

% a single '%' is expected in the input at this point; no assignment is done.

d	a decimal integer is expected; the corresponding argument should be an integer pointer.
o	an octal integer is expected; the corresponding argument should be an integer pointer.
x	a hexadecimal integer is expected; the corresponding argument should be an integer pointer.
s	a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating '\0', which will be added. The input field is terminated by a space character or a newline.
c	a character is expected; the corresponding argument should be a character pointer. The normal skip over space characters is suppressed in this case; to read the next non-space character, try '%1s'. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.
[	indicates a string not to be delimited by space characters. The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of characters making up the string. If the first character is not circumflex (^), the input field is all characters until the first character not in the set between the brackets; if the first character after the left bracket is circumflex (^), the input field is all characters until the first character which is in the remaining set of characters between the brackets. The corresponding argument must point to a character array.

The conversion characters **d**, **o** and **x** may be capitalized or preceded by **l** to indicate that a pointer to **long** rather than to **int** is in the argument list.

The *scanf* functions return the number of successfully matched and assigned input items. This can be used to decide how many input items were found. The constant **EOF** is returned upon end of input; note that this is different from 0, which means that no conversion was done; if conversion was intended, it was frustrated by an inappropriate character in the input.

For example, the call

```
int i; char name[50];
scanf("%d%s", &i, name);
```

with the input line

```
25   thompson
```

will assign to *i* the value 25, and *name* will contain '*thompson\0*'. Or,

```
int i; char name[50];
scanf("%2d%*d%[1234567890]", &i, name);
```

with input

```
56 0123 56a72
```

will assign 56 to *i*, skip '0123', and place the string '56\0' in *name*. The next call to *getchar* will return 'a'.

SEE ALSO

getc(2), printf(3)

DIAGNOSTICS

The *scanf* functions return SYSERR on end of input, and a short count for missing or illegal data items.

BUGS

The success of literal matches and suppressed assignments is not directly determinable.

NAME

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, index, rindex - string operations

SYNOPSIS

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

char *strncat(s1, s2, n)
char *s1, *s2;

strcmp(s1, s2)
char *s1, *s2;

strncmp(s1, s2, n)
char *s1, *s2;

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

char *strncpy(s1, s2, n)
char *s1, *s2;

strlen(s)
char *s;

char *index(s, c)
char *s, c;

char *rindex(s, c)
char *s, c;

DESCRIPTION

These functions operate on null-terminated strings. They do not check for overflow of any receiving string.

Strcat appends a copy of string *s2* to the end of string *s1*. *Strncat* copies at most *n* characters. Both return a pointer to the null-terminated result.

Strcmp compares its arguments and returns an integer greater than, equal to, or less than 0, according as *s1* is lexicographically greater than, equal to, or less than *s2*.

Strncmp makes the same comparison but examines at most *n* characters.

Strcpy copies string *s2* to *s1*, stopping after the null character has been moved.

Strncpy copies exactly n characters, truncating or null-padding *s2*; the target may not be null-terminated if the length of *s2* is *n* or more. Both return *s1*.

Strlen returns the number of non-null characters in *s*.

Index (*rindex*) returns a pointer to the first (last) occurrence of character *c* in string *s*, or zero if *c* does not occur in the string.