TechOS Programmers Manual

Lucas Darnell

West Virginia University Institute of Technology

Jared Miller

West Virginia University Institute of Technology

Ben Culkin

West Virginia University Institute of Technology

Overview

Our TechOS is a command-line OS that is being made so that we can get a good perspective on how Operating Systems are made/operate. The current state is still quite simple, with this being only the second module, and having not that many commands implemented. The project is written in C, and we are using Git to keep track of files.

For this final module, we implemented users of three different categories: basic, administrator and root. Basic users can use all of the normal categories, administrators can create and edit basic users, and root can do the same to administrators.

File Summaries

comlist.h
Header file for declaration of command lists, and the operations upon them.

comlist.c
Implementation of command lists, which is where available commands are stored.

command.h
Header file for declaration of the command structure.

command.c
Implementation of the command structure.

commands.h
Header file for the general command list.

commands.c
Implementation of the general command list.

commandstate.c
Implementation of state passed to each command.

datecmds.h
Header file for declaration of date commands.

datecmds.c
Implementation of the date commands.

dispatchcmds.h
Header file for commands for dispatching PCBs.

dispatchcmds.c
Implementation of commands for dispatching PCBs

filecmds.h
Header file for commands for doing things with files.

filecmds.c
Implementation of commands for doing things with files

osstate.h
Header file for declaration of OS state, which is where all the things we want to keep track of get put to cut down on globals.

osstate.c
Implementation of OS state.

pcb.h
Header file for declarations of PCBs (Process Control Blocks).

pcb.c
Implementation of PCBs.

pcbcmds.h
Header file for declaration of PCB commands.

pcbcmds.c
Implementation of PCB commands.

pcbinternals.h
Header file for implementation details on how PCBs and PCB queues work.

pcbinternals.c
Implementation for any PCB internals.

procexecute.h
Header file for running processes.

procexecute.c
Implementation of running processes.

scriptcmds.h
Header file for declaration of script commands.

scriptcmds.c
Implementation of the script commands: script

techos.h
Main header file for the OS. Contains prototypes for the interface, as well as necessary global data.

techos.c
Implemenation of the interface for the OS. Is responsible for getting commands from the user, and then executing them.

test.tsh
A test file for the script command.

usercmds.h
Header file for user commands.

usercmds.c
Implementation of user commands.

users.h
Header file for users.

users.c
Implementation of users.

users.txt
: Storage for user data.

help/blpcb.1
: Manual for the blpcb command.

help/cd.1
: Manual for the cd command.

help/date.1
: Manual for the date command

help/datefmt.1
: Manual for the datefmt command

help/dispatch.1
: Manual for the dispatch command.

help/exit.1
: Manual for the exit command

help/help.1
: Manual for the help command

help/ls.1
: Manual for the ls command.

help/mkadm.1
: Manual for the mkadm command.

help/mkdir.1
: Manual for the mkdir command.

help/mkpcb.1
: Manual for the mkpcb command.

help/mkusr.1
: Manual for the mkusr command.

help/moo.1
: Manual for the moo command.

help/pwd.1
: Manual for the pwd command.

help/rm.1
: Manual for the rm command.

help/rmadm.1
: Manual for the rmadm command.

help/rmdir.1
: Manual for the rmdir command.

help/rmpcb.1
: Manual for the rmpcb command.

help/rmdir.1
: Manual for the rmdir command.

help/rmpcb.1
Manual for the rmpcb command.

help/rmusr.1
Manual for the rmusr command.

help/rspcb.1
Manual for the rspcb command.

help/script.1
Manual for the script command.

help/setdate.1
Manual for the setdate command

help/shpcb.1
Manual for the shpcb.1 command.

help/sppcb.1
Manual for the sppcb command.

help/sspcb.1
Manual for the sspcb command.

help/time.1
Manual for the time command.

help/touch.1
Manual for the touch command.

help/ubpcb.1
Manual for the ubpcb command.

help/version.1
Manual for the version command.

libs/argparser.h
Header file for the command-line argument parser.

libs/argparser_test.c
Implementation of the test for the command-line argument parser.

libs/argparser.c
Implementation of the command-line argument parser.

libs/intern.h
Header file for a string-interning system.

libs/intern.c
Implemenation of the string-interning system.

Data Structures

struct comlist *(File: comlist.h)*

Usage:

Incomplete type for a command list.
A command list is exactly what its name implies, a list of commands, with the ability to get a command by name, or to do something for every command in the list.

struct comlist** * **makecomlist() *(File: comlist.h)*

Usage:

Allocate and initialize a command list.

Return:

A new command list.

void **killcomlist(*struct comlist *)** *(File: comlist.h)*

Usage:

Destroy a command list

Arguments:

- *struct comlist **
  The command list to destroy.

void **addcommand(*struct comlist *, char *, char *, comfun_t)**
(File: comlist.h)

Usage:

Create and add a command to a command list.

Arguments:

- *struct comlist **
  The command list to add the command to.
- *char **
  The name of the command.
- *char **
  A brief description of the command.
- *comfun_t*
  A pointer to the function that handles the command.

***struct command ** getcommand(*struct comlist *, char **)** *(File: comlist.h)*

Usage:

Retrieve a command from a command list. Will return the ***INVALID_COMMAND*** if there is no command by that name in the list.

Arguments:

- *struct comlist **
  The command list to look in.
- *char **
  The name of the command to look for.

Return:

The command if it was found in the list, or INVALID_COMMAND if it wasn't.

***void* foreachcommand(*struct comlist *, void (*comitr)(struct command *)*)** *(File: comlist.h)*

Usage:

Execute a function for every command in the list.

Arguments:

- *struct comlist **
  The command list to get commands from.
- *void (*comitr)(struct command *)*
  The function to execute for every command.

***void* printcommands(*struct comlist *, FILE **)** *(File: comlist.h)*

Usage:

Prints a command list.

Arguments:

- *struct comlist **
  The command list to print to.
- *FILE **
  The stream to print to.

struct comlist *(File: comlist.c)*

Usage:

A list of commands, stored in an array indexed by an interned string (the command name).

Attributes:

- ***struct command*** ****** **commands*:***
  All of the commands in this list, stored in a dynamically allocated array.
- ***int*** **comspace*:***
  The available number of slots in the command array.
- ***int*** **comcount*:***
  The total number of slots in the command array.
- ***struct interntab*** ***** **interncoms*:***
  The intern table that turns command names into indexes into the command array.

typedef int (*comfun_t)(int , char **, char *, struct osstate *) *(File: command.h)*

Usage:

Represents the type of a pointer to a command handler.

Arguments:

- *int* The number of args the command takes.
- *char ***
  An array of command line arguments.
- *char **
  The entire command line, as the user entered it.
- *struct osstate **
  The current OS state.

Return:

The status of the command. Should be:

- Zero if the command succeded.
- Positive if the command failed in a non-fatal manner.
- Negative if the command failed in a fatal manner.

enum commandtype *(File: command.h)*

Usage:

Represents the different types of command that exist.

Attributes:

- **CT_NORMAL*:***
  A normal command, who is represented by a struct command.

struct command *(File: command.h)*

Usage:

The core data structure that represents an executable command.

Attributes:

- ***enum commandtype* type*:***
  The type of the command. For this struct, should be CT_NORMAL.
- ***char** * **name*:***
  The name of the command.
- ***char** * **brief*:***
  A brief description of the command.
- ***comfun_t* comfun*:***
  The handler that executes the command.

DECLCOM(name) *(File: command.h)*

Usage:

Declare the prototype for a command handler.
Declares a *comfun_t* that is given the name of the command, prefixed with handle_.

HANDLECOM(name) *(File: command.h)*

Usage:

Declare the implementation for a command handler.

***int* checkhelpargs(*int, char **, char *, struct osstate *)** *(File: command.h)*

Usage:

Check the arguments of a command that only takes the '-h/--help' argument to see if it got it as an argument.

Arguments:

- *int* The number of command line arguments.
- *char ***
  An array of the command line arguments.
- *char **
  The command line, just as the user input it.
- *struct osstate **
  The state of the OS.

Return:

Always returns zero.

typedef int (*comfun_t)(int , char **, char *, struct osstate *) *(File: command.h)*

Usage:

Represents the type of a pointer to a command handler.

Arguments:

- *int* The number of args the command takes.
- *char ***
  An array of command line arguments.
- *char **
  The entire command line, as the user entered it.
- *struct osstate **
  The current OS state.

Return:

The status of the command. Should be:

- Zero if the command succeded.
- Positive if the command failed in a non-fatal manner.
- Negative if the command failed in a fatal manner.

struct command *(File: command.h)*

Usage:

The core data structure that represents an executable command.

Attributes:

- ***char** * **name:***
  The name of the command.
- ***char** * **brief:***
  A brief description of the command.
- ***comfun_t* comfun:***
  The handler that executes the command.

DECLCOM(name) *(File: command.h)*

Usage:

Declare the prototype for a command handler.
Declares a *comfun_t* that is given the name of the command, prefixed with handle_.

HANDLECOM(name) *(File: command.h)*

Usage:

Declare the implementation for a command handler.

***int* checkhelpargs(*int, char **, char *, struct osstate **)** *(File: command.h)*

Usage:

Check the arguments of a command that only takes the '-h/--help' argument to see if it got it as an argument.

Arguments:

- *int* The number of command line arguments.
- *char ***
  An array of the command line arguments.
- *char **
  The command line, just as the user input it.
- *struct osstate **
  The state of the OS.

Return:

Always returns zero.

struct command** * **makecommand(char *, char *, comfun_t)* *(File: command.h)*

Usage:

Create a new normal (CT_NORMAL) command.

Arguments:

- *char **
  The name of the command,
- *char **
  A brief description of the command.
- *comfun_t*
  The handler function for the command.

Return:

A new normal command

void** **killcommand(struct command *)* *(File: command.h)*

Destroy a normal (CT_NORMAL) command.

Arguments:

- *struct command **
  The command to destroy.

***void** **initcoms()** *(File: commands.h)*

Usage:

Initialize global state for commands.

void** **addcommands(struct comlist *)* *(File: commands.h)*

Usage:

Add all of the default commands to a command list.

Arguments:

- *struct comlist **
  The list to add the commands to.

***void** **disposecoms()** *(File: commands.h)*

Usage:

Cleanup global state for commands.

DECLCOM(exit) *(File: commands.h)*

Usage:

Implement the **exit** command.

DECLCOM(version) *(File: commands.h)*

Usage:
Implement the **version** command.

DECLCOM(help) *(File: commands.h)*

Usage:
Implement the **help** command.

DECLCOM(moo) *(File: commands.h)*

Usage:
Implement the **moo** command.

DECLCOM(date) *(File: datecmds.h)*

Usage:
Implement the **date** command.

DECLCOM(datefmt) *(File: datecmds.h)*

Usage:
Implement the **datefmt** command.

DECLCOM(setdate) *(File: datecmds.h)*

Usage:
Implement the **setdate** command.

DECLCOM(time) *(File: datecmds.h)*

Usage:
Implement the **time** command.

DECLCOM(dispatch) *(File: dispatchcmds.h)*

Usage:
Implement the **dispatch** command.

int* dodispatch(*struct osstate *, int) *(File: dispatchcmds.c)*

Usage:
Do the actual work of dispatching processes.

Arguments:

- *struct osstate **
  The state of the OS.
- *int* The level of verbosity, or the amount of output to generate. Should be 0, 1 or 2, with higher numbers giving more output.

Return:
The return status of the command

DECLCOM(ls) *(File: filecmds.h)*

Usage:
Implement the **ls** command.

DECLCOM(cd) *(File: filecmds.h)*

Usage:
Implement the **cd** command.

DECLCOM(mkdir) *(File: filecmds.h)*

Usage:
Implement the **mkdir** command.

DECLCOM(rmdir) *(File: filecmds.h)*

Usage:
Implement the **rmdir** command.

DECLCOM(touch) *(File: filecmds.h)*

Usage:
Implement the **touch** command.

DECLCOM(rm) *(File: filecmds.h)*

Usage:
Implement the **rm** command.

struct cliargs *(File: libs/argparser.h)*

Usage:

Contains the command-line arguments parsed from an input line.

Attributes:

- ***int* argc*:***
  The number of command-line arguments in place.
- ***char ** argv*:***
  The array of the command-line arguments. Is an array of *argc* strings.

***struct cliargs* parseargs(*char **)** *(File: libs/argparser.h)*

Usage:

Parse a set of command-line arguments from the given string. The main thing that this function does that using *strtok* doesn't, is that this will properly group together things that are inside single/double quoted strings as a single argument.

Arguments:

- *char **
  The line to parse command-line arguments from. This string is mutated.

Return:

The parsed set of command-line arguments.

***char ** parsestr(*char *, char, char ***)** *(File: libs/argparser.c)*

Usage:

Parse a string from a input line.

Arguments:

- *char **
  The initial part of the string, up until the first space.
- *char*
  The character that should end this string.
- *char ***
  The string being parsed for arguments.

typedef *int* internkey *(File: libs/intern.h)*

Usage:

The type of a key into an intern table. Anything less than or equal to zero is a invalid key.

struct interntab *(File: libs/intern.h)*

Usage:

Opaque state-type for a intern table.
An intern table is a container for a set of interned strings. The primary purpose for this is to map a string to a small integer key, for easy indexing of data structures and string equality checking.

struct interntab** * **makeinterntab() *(File: libs/intern.h)*

Usage:

Create a new intern table

Return:

The new intern table

void **killinterntab(*struct interntab **)** *(File: libs/intern.h)*

Usage:

Destroy an intern table.

Arguments:

- *struct interntab **
  The intern table to destroy

internkey **internstring(*struct interntab **, *const char **)** *(File: libs/intern.h)*

Usage:

Enter a string (intern it) into a intern table.

Arguments:

- *struct interntab **
  The intern table to enter the string into.
- *const char **
  The string to intern.

Return:

The key corresponding to the string. This may or may not be a new key, depending on whether the string was interned already or not.

***internkey* lookupstring(*struct interntab *, const char *)** *(File: libs/intern.h)*

Usage:

Look up the key for a string in the intern table.

Arguments:

- *struct interntab*
  The intern table to enter the string into.
- *const char **
  The string to look up the intern key for.

Return:

The intern key corresponding to the strings, or the special key SIINVALID if the provided string hasn't been interned.

***const char ** lookupkey(*struct interntab *, internkey)** *(File: libs/intern.h)*

Usage:

Look up the string for a key in the intern table.

Arguments:

- *struct interntab*
  The intern table to enter the string into.
- *internkey*
  The intern key to look up the string for.

Return:

The string that has the given intern key, or the NULL pointer if no string has been interned to that key.

typedef void (*tableitr)(const char *, internkey, void * *(File: libs/intern.h)*

Usage:

The type of an iterator over an intern table. Takes the following parameters:

- The string in the table.
- The key corresponding to the string.
- A piece of user-supplied data.

void* foreachintern(*struct interntab *, tableitr, void *)

(File: libs/intern.h)

Usage:

Execute an action over every K-V pair in a table.

Arguments:

- *struct interntab **
  The intern table to iterate over.
- *tableitr*
  The iterator to execute.
- *void **
  The data to pass to the iterator.

struct internlist *(File: libs/intern.h)*

Usage:

Incomplete type for an intern-list, which is a map that uses interned strings as keys.

struct internlist ** makeinternlist(*int, void (*pfDestroy)(void *))

(File: libs/intern.h)

Usage:

Create an intern list.

Arguments:

- *int* The initial capacity of the list.
- *void (*pfDestroy)(void *)*
  The function to call when an item is deleted from the list.

Return:

The new intern list.

void* killinternlist(*struct internlist *) *(File: libs/intern.h)*

Usage:

Destroy and delete an intern list.

Arguments:

- *struct internlist **
  The intern list to destroy.

void* putinternlist(*struct internlist *, char *, void *)

(File: libs/intern.h)

Usage:

Insert an item into an intern list.

Arguments:

- *struct internlist **
  The intern list to add to.
- *char **
  The key to index the data by.
- *void **
  The data to attach to the key.

void ** getinternlist(*struct internlist *, char *) *(File: libs/intern.h)*

Usage:

Get an item from an intern list. Returns NULL if there is no item for that key.

Arguments:

- *struct internlist **
  The intern list to get the item from.
- *char **
  The key to lookup a value for.

Return:

The item in the list, or NULL if there is no item for that key.

void* deleteinternlist(*struct internlist *, char *) *(File: libs/intern.h)*

Usage:

Delete an item from an intern list. Does nothing if there is no item attached to that key.

Arguments:

- *struct internlist **
  The intern list to remove the item from.
- *char **
  The key to remove a value for.

***int* containsinternlist(*struct internlist *, char **)** *(File: libs/intern.h)*

Usage:

Check if an item is in an intern list. Returns 0 if the item is not in the list, 1 if it is in the list.

Arguments:

- *struct internlist **
  The list to check for membership in.
- *char **
  The key to check for.

Return:

1 if the item is in the list, 0 if it isn't.

typedef void (*internlistitr)(char *, void *, void *) *(File: libs/intern.h)*

Usage:

Type of an iterator for an intern list. Takes the following arguments:

- The string key of the item.
- The value of the item.
- A piece of user supplied data.

***void* foreachinternlist(*struct internlist *, internlistitr, void **)**

(File: libs/intern.h)

Usage:

Execute a function for every key-value pair in a list.

Arguments:

- *struct internlist **
  The list to iterate over.
- *internlistitr*
  The function to execute for each key-value pair.
- *void **
  A piece of data to pass to each method.

***unsigned long* hashstring(*const char **)** *(File: libs/intern.c)*

Usage:

Convert a string to a hashcode. This uses the DJB2K33 algorithm, which has been shown to work well for a wide variety of strings, and not be terribly expensive to compute.

Arguments:

- *const char **
  The string to hash.

Return:

The hashcode for the string.

unsigned long* hashkey(*const int) *(File: libs/intern.c)*

Usage:

Convert an integer to a hashcode.

Arguments:

- *const int*
  The integer to hash.

Return:

The hashcode for the integer.

struct bucket *(File: libs/intern.c)*

Usage:

A bucket in one of the two hash-tables the intern table manages.
This bucket is actually a circular linked list of buckets.

Attributes:

- ***char ** val:**
  The string in this bucket.
- ***internkey* key:**
  The intern key for this bucket.
- ***struct bucket ** next:**
  Pointer to the next bucket in the chain.
- ***struct bucket ** prev:**
  Pointer to the previous bucket in the chain.

struct interntab *(File: libs/intern.c)*

Usage:

The implementation of the opaque type in *libs/intern.h*.
Contains the value of the next intern key that will be assigned, and two hashtables. One is keyed off of the interned strings, and the other is keyed off of the keys for those strings. This is so we can get efficent lookup of both keys from strings, as well as strings from their corresponding keys.

Attributes:

- ***nextkey :***
  The intern key that will be assigned to the next string interned into the table.
- ***strings :***
  A hash-table keyed off of the string values, for looking up keys based off of strings.
- ***keys :***
  A hash-table keyed off of the key values, for looking up strings based off of keys.

***void* addbucket(*struct bucket *)** *(File: libs/intern.c)*

Usage:

Add a bucket to a bucket chain.

Arguments:

- *struct bucket **
  The chain to add a bucket to.

struct internlist *(File: libs/intern.c)*

Usage:

The implementation for the opaque data type of the same name in libs/intern.h. A simple array-based mapping from strings to void pointers.

Attributes:

- ***paData :***
  The actual data storage for the array.
- ***dataspace :***
  The amount of space available in the data array.
- ***datausage :***
  The amount of space used in the data array.
- ***ptKeys :***
  The mapping from string keys to indexes into the array.
- ***void (*pfDestroy)(void *) :***
  The function to call on data items that are being deleted.

struct listitrdata *(File: libs/intern.c)*

Usage:

Storage for data used in dointernlistitr.

Attributes:

- ***plList :***
  The list being iterated over.
- ***pfFunc :***
  The function to call for each pair.
- ***pvData :***
  The data to pass to pfFunc.

void* dointernlistitr(*const char *, internkey, void *)

(File: libs/intern.c)

Usage:

Internal function for using foreachintern to implement foreachintern-list.

Arguments:

- *const char **
  The name of the entry in the intern list.
- *internkey*
  The key of the entry in the intern list.
- *void **
  The listitrdata we use for data storage.

struct commandstate *(File: osstate.h)*

Usage:

Forward declaration of the state of commands, stored elsewhere.

struct osstate *(File: osstate.h)*

Usage:

Contains general state for things the OS requires.

Attributes:

- ***char** * **in_datefmt:***
  The current input format for dates.
- ***char** * **out_datefmt:***
  The current output format for dates.
- ***char** * **time_datefmt:***
  The current output format for times.
- ***struct tm** * **datetime:***
  The current date/time.
- ***FILE** * **strem:***
  The source for reading input from.
- ***FILE** * **output:***
  The source for writing output to.
- ***struct pcbstate** * **pPCBstat:***
  The state needed for PCBs.
- ***struct commandstate** * **pComstate:***
  The state of commands.
- ***int** **fWorkingDir:***
  File descriptor for the base working dir. of the OS.
- ***struct user** * **puCurrent:***
  The current logged in user.
- ***struct userdb** * **pdUsers:***
  The database of users.

struct osstate** * **makeosstate() *(File: osstate.h)*

Usage:

Allocate and initialize a OS state struct.

void **killosstate(*struct osstate **)** *(File: osstate.h)*

Usage:

Deinitialize and deallocate an OS state struct.

enum pcbclass *(File: pcb.h)*

Usage:
Represents the classification of a PCB.

Attributes:

- **PCB_SYSTEM*:***
  This PCB represents a system process.
- **PCB_APPLICATION*:***
  This PCB represents a user process.

enum pcbstatus *(File: pcb.h)*

Usage:
The run status of a PCB.

Attributes:

- **PCB_BLOCKED*:***
  The PCB is blocked, waiting for something.
- **PCB_READY*:***
  The PCB is ready to run.
- **PCB_RUNNING*:***
  The PCB is currently running.

enum pcbsusp *(File: pcb.h)*

Usage:
The suspension status of a PCB.

Attributes:

- **PCB_SUSPENDED*:***
  This PCB is currently suspended.
- **PCB_FREE*:***
  This PCB is currently not suspended.

struct pcb *(File: pcb.h)*

Represents a running process.

Attributes:

- ***int* id*:***
  The unique numeric ID for the process.
- ***internkey* kName*:***
  The name for the process. Stored as a interned string, for fast equality checking and space savings.
- ***enum pcbclass* class*:***
  The classification of this process (System/Application).
- ***int* priority*:***
  The priority of this process, on a scale from 0 to 9, with higher numbers being higher priority.
- ***enum pcbstatus* status*:***
  Indicates whether the PCB is running, ready to run, or blocked.
- ***enum pcbsusp* susp*:***
  Indicates whether the PCB is suspended or not.
- ***internkey* kImage*:***
  The key for the image of this process.
- ***int* offset*:***
  The current execution offset of the process.
- ***struct pcb* * pNext*:***
  The next process in the queue this PCB is in.
- ***struct pcb* * pPrev*:***
  The previous process in the queue this PCB is in.

(File:)

Usage:

Forward declaration for queues of PCBs.

Usage:

Forward declaration for the container of all of the PCB states.

struct pcb * **makepcb(*struct pcbstate *, char *, char *, enum pcbclass, int*)**

(File: pcb.h)

Usage:

Allocate/initialize a new PCB.

Arguments:

- *struct pcbstate **
  The current state for all PCBs.
- *char **
  The name of the PCB.
- *char **
  The path to the process image for the PCB.
- *enum pcbclass*
  The classification of the PCB.
- *int* The priority of the PCB.

Return:

An allocated and initialized PCB.

void **killpcb(*struct pcb **)** *(File: pcb.h)*

Usage:

Deinitialize and deallocate a PCB.

Arguments:

- *struct pcb **
  The PCB to deallocate.

struct pcb * **findpcbnum(*struct pcbstate *, int*)** *(File: pcb.h)*

Usage:

Find a PCB by its numeric ID in a set of queues.

Arguments:

- *struct pcbstate **
  The set of queues to search in.
- *int* The process number to look for.

Return:

The PCB with the corresponding number, or NULL if no process with that number exists.

struct pcb** * **findpcbname(struct pcbstate *, char ***)*** *(File: pcb.h)*

Usage:

Find a PCB by its name in a set of queues.

NOTE:

Since process names are not guaranteed to be unique, the queues are searched in the following order, with the first PCB with a matching name being returned.

1. Ready
2. Blocked
3. Suspended Ready
4. Suspended Blocked

Arguments:

- *struct pcbstate **
  The set of queues to look in.
- *char **
  The process name to look for.

Return:

The first process with the given name following the above search order, or NULL if no process by that name exists.

enum pcberror *(File: pcb.h)*

Usage:

The errors possible for ***insertpcb()***

Attributes:

- **PCBSUCCESS*:***
  No error occured.
- **PCBINVSUSP*:***
  The provided PCB has an invalid suspension status
- **PCBINVSTAT*:***
  The provided PCB has an invalid run status
- **PCBRUNNING*:***
  The provided PCB is currently running.
- **PCBINQUEUE*:***
  The provided PCB is already in a queue.

***enum pcberror* insertpcb(*struct pcbstate *, struct pcb **)** *(File: pcb.h)*

Usage:
Insert a PCB into the proper queue, based off of its status.

Arguments:

- *struct pcbstate **
  The set of queues to insert the PCB into.
- *struct pcb **
  The PCB to insert into a queue.

Return:
The status of the insertion operation.

***void* removepcb(*struct pcbstate *, struct pcb **)** *(File: pcb.h)*

Usage:
Remove a PCB from the queue it is currently in.

NOTE:
Attempting to remove a PCB that is not in a queue from a queue may cause various degrees of weird things to happen. Don't do it.

Arguments:

- *struct pcbstate **
  The set of queues to remove a PCB from.
- *struct pcb **
  The PCB to remove from the queues.

***int* candispatch(*struct pcbstate **)** *(File: pcb.h)*

Usage:
Check the number of currently dispatchable summons.

Arguments:

- *struct pcbstate **
  The state to look for PCBs in.

Return:
The number of processes that can be dispatched.

***struct pcb ** getdisppcb(*struct pcbstate **)** *(File: pcb.h)*

Usage:

Get a process to dispatch, which is the process with the highest priority that is not suspended.

Arguments:

- *struct pcbstate **
  The state to look for PCBs in.

Return:

The dispatchable process, or NULL if there are no dispatchable processes.

***struct pcb ** queuefindpcbnum(*struct pcbqueue *, int)** *(File: pcb.c)*

Usage:

Find something by process number in a PCB queue.

Arguments:

- *struct pcbqueue **
  The queue to look in.
- *int* The process number to look for.

Return:

The PCB with that process number, or null if no such PCB exists.

***struct pcb ** queuefindpcbname(*struct pcbqueue *, int)** *(File: pcb.c)*

Usage:

Find something by process name in a PCB queue.

NOTE:

Since process names aren't guaranteed to be unique, will give the first PCB with that name.

Arguments:

- *struct pcbqueue **
  The queue to look in.
- *int* The string key for the process name to look for.

Return:

The first PCB with that process name, or null if no such PCB exists.

***int* fillqueue(*struct pcbqueue *, struct pcb **)** *(File: pcb.c)*

Usage:
Fill a PCB queue if it is empty.

Arguments:

- *struct pcbqueue **
  The queue to attempt to fill.
- *struct pcb **
  The PCB to attempt to fill with.

Return:
1 if the queue was succesfully filled, 0 if it already had something in it.

***void* fifoinsertpcb(*struct pcbqueue **)** *(File: struct pcb *)*

Usage:
Insert a PCB into a FIFO queue.

Arguments:

- *struct pcbqueue **
  The queue to insert into.
- *struct pcb **
  The PCB to insert into the queue.

***void* filoinsertpcb(*struct pcbqueue **)** *(File: struct pcb *)*

Usage:
Insert a PCB into a FILO queue.

Arguments:

- *struct pcbqueue **
  The queue to insert into.
- *struct pcb **
  The PCB to insert into the queue.

***void* priorinsertpcb(*struct pcbqueue **)** *(File: struct pcb *)*

Usage:
Insert a PCB into a priority queue.

Arguments:

- *struct pcbqueue **
  The queue to insert into.
- *struct pcb **
  The PCB to insert into the queue.

DECLCOM(mkpcb) *(File: pcbcmds.h)*

Usage:
Implement the **mkpcb** command.

DECLCOM(rmpcb) *(File: pcbcmds.h)*

Usage:
Implement the **rmpcb** command.

DECLCOM(blpcb) *(File: pcbcmds.h)*

Usage:
Implement the **blpcb** command.

DECLCOM(ubpcb) *(File: pcbcmds.h)*

Usage:
Implement the **ubpcb** command.

DECLCOM(sspcb) *(File: pcbcmds.h)*

Usage:
Implement the **sspcb** command.

DECLCOM(rspcb) *(File: pcbcmds.h)*

Usage:
Implement the **rspcb** command.

DECLCOM(sppcb) *(File: pcbcmds.h)*

Usage:
Implement the **sppcb** command.

DECLCOM(shpcb) *(File: pcbcmds.h)*

Usage:
Implement the **shpcb** command.

***void* printpcb(*struct pcb *, void **)** *(File: pcbcmds.c)*

Usage:

Print a PCB.

Arguments:

- *struct pcb **
  The PCB to print.
- *void **
  The state parameter. Must be valid to cast to a ‘‘struct osstate *’’

***int* executeimage(*FILE *, int*)** *(File: procexecute.h)*

Usage:

Execute an image from a file.

Arguments:

- *FILE **

enum queuetype *(File: pcbinternals.h)*

Usage:

Represents the various types of PCB queues that exist.

Attributes:

- **QT_NORMAL*:***
  Represents a normal, FIFO PCB queue. The corresponding struct is `struct pcbqueue`.
- **QT_PRIORITY*:***
  Represents a priority FIFO PCB queue. The corresponding struct is `struct pcbqueueprior.`

struct pcbqueue *(File: pcbinternals.h)*

Usage:

Represents a circular, doubly-linked list of PCBs

Attributes:

- ***enum pcbtype* type*:***
  The type of this PCB. Indicates the actual struct type of this PCB, which may not be struct pcbqueue. Should be QT_NORMAL if this is actually a `struct pcbqueue`.
- ***int* nprocs*:***
  The number of processes in this queue.
- ***struct pcb* * pHead*:***
  The head of the queue.

struct pcbqueueprior *(File: pcbinternals.h)*

Usage:

Represents a priority queue of PCBs.

Attributes:

- ***enum queuetype* type*:***
  The type of queue this is. Should always be QT_PRIOR.
- ***int* nprocs*:***
  The number of processes in this queue.
- ***struct pcbqueue* ** apqQueues*:***
  The list of queues: one for each priority level.

struct pcbstate *(File: pcbinternals.h)*

Usage:

Contains all of the state needed for PCBs

Attributes:

- ***struct interntab* * ptPCBNames*:***
  The table where process names are stored.
- ***int* nProcid*:***
  The next process ID to be assigned.
- ***struct pcbqueue* * pqReady*:***
  The queue of ready PCBs.
- ***struct pcbqueue* * pqBlocked*:***
  The queue of blocked PCBs.
- ***struct pcbqueue* * pqsReady*:***
  The queue of suspended ready PCBs.
- ***struct pcbqueue* * pqsBlocked*:***
  The queue of suspended blocked PCBs.

***void* foreachpcb(*struct pcbqueue *, void (*pcbitr)(struct pcb *, void *), void *)**
(File: pcbinternals.h)

Usage:

Execute a function once for each PCB in a queue.

Arguments:

- *struct pcbqueue **
  The queue of PCBs to process.
- *void (*pcbitr)(struct pcb *, void *)*
  The function to execute for each PCB. Takes an extra state parameter.
- *void **
  The state parameter to pass to the function.

***void* comhan(*struct osstate **)** *(File: techos.h)*

Usage:

The main command handler.

Loops reading commands and handling them until either an EOF is reached, an exit command is executed, or a command fails in a fatal manner.

Arguments:

- *struct osstate **
  The state of the OS during command execution.

***int* handleline(*struct osstate *, char **)** *(File: techos.h)*

Usage:

Executes a command from a line of input.

Arguments:

- *struct osstate **
  The state of the OS for the line.
- *char **
  The line to execute.

Return:

The return status of the command. This is zero for success, positive for non-fatal failure, and negative for fatal failure.

***int* execcom(*struct command *, struct cliargs, char *, struct osstate **)**
(File: techos.h)

Usage:

Execute a command, given its arguments.

Arguments:

- *struct command **
  The command to execute.
- *struct cliargs*
  The arguments to the command.
- *char **
  The command-line as the user entered it.
- *struct osstate **
  The state of the OS for the command.

Return:

The return status of the command, the meaning of which is documented above.

***int* main()** *(File: techos.c)*

Usage:

Entrance point for TechOS. Setups state, prints a greeting, invokes the command handler, then cleanups up the result and exits.

Return:

The status code for the program

Global Variables

struct command INVALID_COMMAND *(File: command.h)*

Usage:
The command that is recognized as an invalid command.

char * defin_datefmt *(File: osstate.h)*

Usage:
The default input date format.

char * deftime_datefmt *(File: osstate.h)*

Usage:
The default time output format.

char * *(File: defout_datefmt)*

The default date output format.

PCB_MINPRIOR *(File: pcb.h)*

Usage:
The minimum priority value for a PCB.

PCB_MAXPRIOR *(File: pcb.h)*

Usage:
The maximum priority value for a PCB.

struct comlist *all_commands *(File: techos.h)*

Usage:
List of all currently registered commands.

int major_ver *(File: techos.h)*

Usage:
The major version of TechOS.

int minor_ver *(File: techos.h)*

Usage:
The minor version of TechOS.

Cross Reference

main *(File: techos.c)*

Calls:
initcoms(), makeosstate(), makecomlist(), addcommands(), comhan(), killosstate() disposecoms(),

Called By:
None

addcommand *(File: comlist.c)*

Calls:
internstring()

Called By:
addcommands()

addcommands *(File: commands.c)*

Calls:
addcommand()

Called By:
main()

checkhelpargs *(File: command.c)*

Called By:
handle_exit(), handle_version(), handle_help(), handle_date(), handle_time(), handle_setdate()

comhan *(File: techos.c)*

Calls:
handleline()

Called By:
main()

disposecoms *(File: commands.c)*

Calls:
Nothing

Called By:
main()

execcom *(File: commands.c)*

Calls:
All the handle_*() functions through com.comfun

Called By:
handleline()

findpcbname *(File: pcb.c)*

Calls:
Nothing

Called By:
handle_rmpcb(), handle_blpcb(), handle_ubpcb(), handle_sspcb(), handle_rspcb(), handle_shpcb()

findpcbnum *(File: pcb.c)*

Calls:
Nothing

Called By:
handle_rmpcb(), handle_blpcb(), handle_shpcb()

foreachcommand *(File: comlist.c)*

Calls:
Nothing

Called By:
Nothing

foreachpcb *(File: pcbinternals.c)*

Calls:
Nothing

Called By:
handle_shpcb()

getcommand *(File: comlist.c)*

Calls:
getcommand()

Called By:
handleline()

handle_* *(File: *cmds.c)*

Called By:
execcom() through com.comfun

handleline *(File: techos.c)*

Calls:
parseargs(), getcommand(), execcom()

Called By:
handle_script(), comhan()

initcoms *(File: commands.c)*

Calls:
Nothing

Called By:
main()

insertpcb *(File: pcb.c)*

Calls:
priorinsertpcb(), fifoinsertpcb()

Called By:
handle_mkpcb(), handle_blpcb(), handle_ubpcb(), handle_sspcb(), handle_rspcb()

internstring *(File: libs/intern.c)*

Calls:
lookupstring(), hashstring(), hashkey(), addbucket()

Called By:
addcommand()

killcomlist *(File: comlist.c)*

Calls:
killinterntab()

Called By:
Nothing

killinterntab *(File: libs/intern.c)*

Called By:
killcomlist()

killosstate *(File: osstate.c)*

Called By:
main()

killpcb *(File: pcb.c)*

Called By:
handle_rmpcb()

lookupkey *(File: libs/intern.c)*

Calls:
hashkey()

Called By:
printpcb()

lookupstring *(File: libs/intern.c)*

Calls:
hashstring()

Called By:
lookupstrings()

makecomlist *(File: comlist.c)*

Calls:
Nothing

Called By:
main()

makeinterntab *(File: libs/intern.c)*

Calls:
Nothing

Called By:
Nothing

makeosstate *(File: osstate.c)*

Calls:
Nothing

Called By:
main()

makepcb *(File: pcb.c)*

Calls:
Nothing

Called By:
handle_mkpcb()

parseargs *(File: libs/argparser.c)*

Calls:
Nothing

Called by:
handleline()

printcommands *(File: comlist.c)*

Calls:
Nothing

Called By:
printcommands()

removepcb *(File: pcb.c)*

Calls:
Nothing

Called By:
hadle_rmpcb(), handle_blpcb(), handle_ubpcb(), handle_sspcb(), handle_rspcb()

Table of Contents