



User Manual

Version - R6

Table of Commands

Basic Use Instructions - 3

R1 - User Input Module - 4

1. help - 4
2. version - 5
3. time - 6
4. date - 7
5. crewmate - 8
6. history - 9
7. clear - 10
8. shutdown - 11

R2 - PCB Operations Module - 13

1. pcb - 13
2. pcb create [name] - 13
3. pcb delete [name] - 14
4. pcb block [name] - 14
5. pcb unblock [name] - 15
6. pcb suspend [name] - 16
7. pcb resume [name] - 16
8. pcb priority [name] [0-9] - 17
9. pcb show [name] - 17
10. pcb all - 18
11. pcb ready - 18
12. pcb blocked - 19

R3 - Dispatch Module - 20

1. yield - 20
2. loadr3 - 20
3. pcb resumeall - 21

R4 - Dispatch Module - 23

1. Alarm - 23

R5 - Memory Management Module - 24

1. mem showallocated - 24
2. mem showfree - 24

R6 - Interrupt Driven I/O - 25

Basic Use Instructions

amogOS is a text-based operating system, which allows users to input different commands to perform various functions. To enter a command, simply type the command into the prompt and press enter to process the command. amogOS commands will be read in appropriately, regardless of any whitespace present in the input commands. As of version R4, amogOS commands are case sensitive, and are all to be input to the command line in lowercase. See the rest of the document for details on the different commands available in amogOS.

R1 - User Input Module

help:

The *help* command displays the different commands that can be run in the console.

```
The help command helps you get information about other commands.  
Currently available commands:  
- time  
- date  
- clear  
- version  
- history  
- shutdown  
- crewmate  
Use "help [command]" for more info on a particular command.
```

The output from running the help command with no parameters

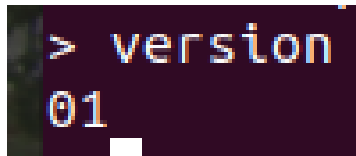
The *help* command can be run with a single parameter, which is the command to show more information for. This is done by typing the name of the command you would like to learn more about after the word *help* in the command line. For example, to display more information about how to use the *time* command, type "*help time*" into the command line)

```
> help time  
Time can be used 3 different ways:  
    time - gets and outputs the system time  
    time get - gets and outputs the system time  
    time set [hour]:[minute]:[second] - sets system time to the specified time
```

**The output displayed to the screen by running the command:
*help time***

version:

The *version* command displays the current version of amogOS that is running in the console. The *version* command does not take any parameters. As of the completion of the first release, this command will display the version as version 01.

A screenshot of a terminal window with a dark background. The prompt character is a green greater-than sign (>). The command 'version' is entered in a light blue font. The output '01' is displayed in the same light blue font on the line below the command.

```
> version
01
```

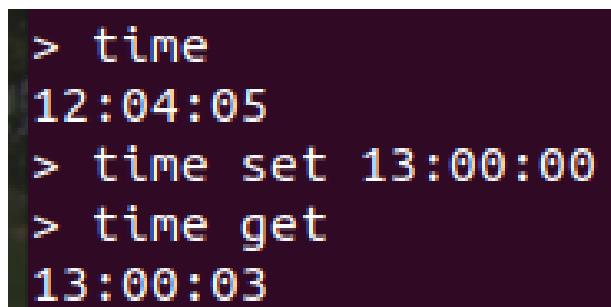
The *version* command, run on version 01 of amogOS

***time*:**

The *time* command is used for displaying and changing the time stored by amogOS. It can be used in a few different ways:

- *time* (with no parameters)
 - Running the *time* command without any parameters will display the current time to the screen
- *time get*
 - Running *time get* will display the current time to the screen. It works the same as *time* with no parameters.
- *time set* [hour]:[minute]:[second]
 - The *time set* command allows the time stored by amogOS to be updated. This is done by running the command, along with the time to update it to in the format [hour]:[minute]:[second]. Note that time in amogOS is stored and displayed using 24-hour time (military time), so it will accept hours from 00 to 23, minutes from 00 to 59, and seconds from 00 to 59.

Note that all use information can be found in the terminal using the *help* command.



```
> time
12:04:05
> time set 13:00:00
> time get
13:00:03
```

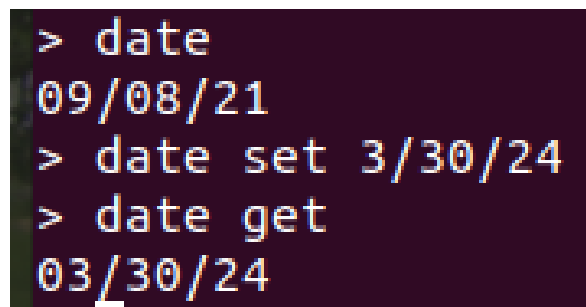
An example usage of the *time* command to update the time to 13:00 (1PM)

date:

The *date* command is used for displaying and changing the date stored by amogOS. It can be used in a few different ways:

- *date* (with no parameters)
 - Running the *date* command without any parameters will display the current date to the screen
- *date get*
 - Running *date get* will display the current date to the screen. It works the same as *date* with no parameters.
- *date set* [month]/[day]/[year]
 - The *date set* command allows the date stored by amogOS to be updated. This is done by running the command, along with the date to update it to in the format [month]/[day]/[year]. Note that the month can be from 1-12, the day can be from 1-31, and the year can be 00-99 (amogOS only displays the last 2 digits of the current year)

Note that all use information can be found in the terminal using the *help* command.



```
> date
09/08/21
> date set 3/30/24
> date get
03/30/24
```

An example usage of the *date* command to update the date to 3/30/24 (March 30, 2024)

***crewmate*:**

The *crewmate* command allows for the selection of a “crewmate” to theme the terminal after. The command is run as “*crewmate*”, with no additional parameters. It will then display an output as seen below. (Theming after a crewmate in amogOS means changing the color of all text in the console).

```
> crewmate
Select a color for your crewmate:
white, red, green, blue, purple, yellow, orange, brown, pink, gray
```

The output of running the *crewmate* command

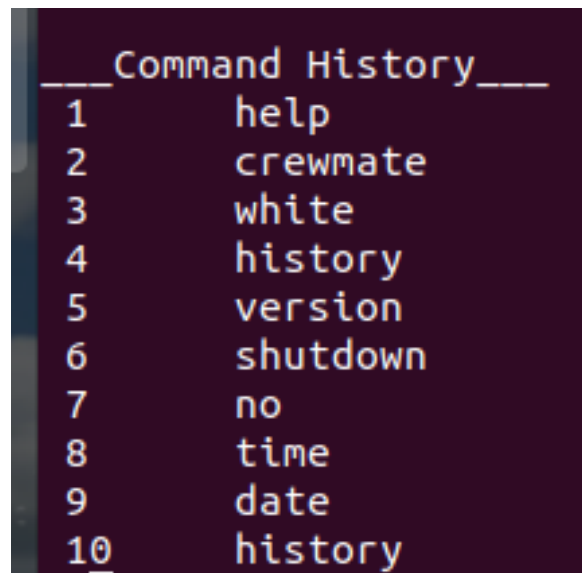
To select a color for the crewmate, type the name of the color to change to. Doing so will change the color of all text printed out in amogOS until the *crewmate* command is run again. Note that entering anything other than the name of one of the colors here will cause the terminal to return to the default command prompt.

```
> crewmate
Select a color for your crewmate:
white, red, green, blue, purple, yellow, orange, brown, pink, gray
> green
Selected GREEN
> help
The help command helps you get information about other commands.
Currently available commands:
- time
- date
- clear
- version
- history
- shutdown
- crewmate
Use "help [command]" for more info on a particular command.
>
```

The *crewmate* command being used to change the color of all text to green

history:

The *history* command will display the 10 most recent commands the user has run in the current session. The *history* command does not take any parameters. The history of commands can also be accessed by using the up and down arrow keys, with the up arrow showing commands farther in the history, and the down arrow showing commands more recent in the history.

A screenshot of a terminal window showing the output of the 'history' command. The output is a list of 10 commands, numbered 1 through 10, with the command names aligned to the right of the numbers. The terminal has a dark background with light-colored text.

```
__Command History__
1      help
2      crewmate
3      white
4      history
5      version
6      shutdown
7      no
8      time
9      date
10     history
```

An example output of the *history* command

clear:

The *clear* command clears the screen, removing all previously printed text from the screen and returning the prompt to the top of the screen. It does not take any parameters.



The screen of amogOS after running the *clear* command

shutdown:

The *shutdown* command is used to shut down amogOS. The command has no parameters. Upon running the command, a prompt will be displayed to the screen asking the user to confirm their decision to shut the system down. Replying with “yes” will cause the system to shut down, and replying with anything else will return the user to the normal command prompt. Once the system has been shut down, the user can return to the Linux command line by entering the inputs ctrl-a + x.

```
> shutdown
Are you sure you want to shut down? (yes/no)
> yes
```



The *shutdown* command being used to shut off amogOS

R2 - PCB Operations Module

pcb:

The *pcb* command displays the all process to the user onto the screen. The output of the *pcb* command segregates the processes by ready, suspended ready, blocked and suspended block queue.

```
Successfully blocked john  
> pcb  
Ready Queue (2):  
jay (7)  
jason (3)  
  
Suspended Ready Queue (1):  
brian (5)  
  
Blocked Queue (1):  
john (1)  
  
Suspended Blocked Queue (0):  
> █
```

pcb create [name] [0-9]:

This command creates a new process with the name specified by the [name] parameter and a priority, which is a value between 0 and 9 (inclusive). A larger value indicates that a process has a higher priority. By default, the process is placed in the ready queue. The [name] parameter must be between 3-20 characters long. The [name] parameter must be unique, since no two PCBs are allowed to share a name.

```
> pcb create john 3  
PCB Created:  
Name: john  
Class: User  
State: Ready  
Priority: 3  
Prev PCB: brian  
Next PCB: Null  
> █
```

pcb delete [name] :

This command deletes a PCB (process control block) with the given name. The user has to pass in a [name] parameter.

```
the process "jason" is already blocked
> pcb delete john
Successfully deleted "john"
>
```

pcb block [name] :

This command blocks a PCB (process control block) with the given name. The user has to pass in a [name] parameter. The user will notice that the PCB will have been moved to the blocked queue after running this command.

```
the process "jason" is not blocked
> pcb block jason
Successfully blocked "jason"
> pcb
Ready Queue (2):
brian (5)
john (1)

Suspended Ready Queue (1):
jay (7)

Blocked Queue (1):
jason (3)

Suspended Blocked Queue (0):
>
```

```
Suspended Blocked Queue (0):
> pcb block jason
The process "jason" is already blocked
```

pcb unblock [name] :

This command unblocks a PCB (process control block) with the given name. The user has to pass in a [name] parameter. If the user specifies a process that is not blocked, the command line will return a statement that informs the user that the process specified is not currently blocked. If the user correctly specifies a blocked process, they will notice it would have been moved to the ready queue after running the command.

```
> pcb
Ready Queue (2):
brian (5)
john (1)

Suspended Ready Queue (1):
jay (7)

Blocked Queue (1):
jason (3)

Suspended Blocked Queue (0):
> pcb unblock jason
Successfully unblocked "jason"
> pcb
Ready Queue (3):
brian (5)
jason (3)
john (1)

Suspended Ready Queue (1):
jay (7)

Blocked Queue (0):

Suspended Blocked Queue (0):
>
```

```
Suspended Blocked Queue (0):
> pcb unblock jason
The process "jason" is not blocked
>
```

pcb suspend [name] :

This command suspends a PCB (process control block) with the given name. The user has to pass in a [name] parameter. If the user specifies a process that is already suspended, the command line will return a statement that informs the user that the process specified is already suspended. If the user correctly specifies an unsuspended process, they will notice it would have been moved to the suspended queue if it was originally in the ready queue, or the suspended blocked queue if it was originally in the blocked queue after running the command.

```
Suspended Blocked Queue (0):  
> pcb suspend jay  
Successfully suspended "jay"  
> pcb  
Ready Queue (3):  
brian (5)  
jason (3)  
john (1)  
  
Suspended Ready Queue (1):  
jay (7)  
  
Blocked Queue (0):  
  
Suspended Blocked Queue (0):  
> █
```

```
Suspended Blocked Queue (0):  
> pcb suspend jay  
The process "jay" is already suspended  
> █
```


pcb resume [name] :

This command “unsuspends” a PCB (process control block) with the given name. The user has to pass in a [name] parameter. If the user specifies a process that is already in a non-suspended state(i.e. ready or blocked), the command line will return a statement that informs the user that the process specified is not suspended. If the user correctly specifies a suspended process, they will notice it would have been moved to the ready queue, if it was originally in the suspended ready queue or the blocked queue if it was originally in the suspended blocked queue after running the command.

```
Suspended Ready Queue (1):
brian (5)

Blocked Queue (0):

Suspended Blocked Queue (0):
> pcb resume brian
Successfully resumed "brian"
> pcb
Ready Queue (4):
jay (7)
brian (5)
jason (3)
john (1)

Suspended Ready Queue (0):

Blocked Queue (0):

Suspended Blocked Queue (0):
> 
```

```
The process "brian" is already suspended
> pcb resume jay
The process "jay" is not suspended
> 
```

pcb priority [name] [0-9] :

This command changed the priority of a PCB (process control block). The user must specify a valid name with the [name] parameter and a new priority with the [0-9] (priority parameter). If erroneous parameters are given (priority out of bounds or non existent PCB), the system will let the user know.

```
> pcb priority john 4
Updated priority of "john"
>
```

```
> pcb priority jahn 10
Invalid input for PCB command
>
```

pcb show [name]:

This command displays the PCB (process control block) with the given name. The user must specify a valid name with the [name] parameter. If erroneous parameters are given (non existent PCB), the system will let the user know.

```
> pcb show john
Name: john
Class: User
State: Ready
Priority: 4
Prev PCB: brian
Next PCB: Null
>
```

```
> pcb show jahn
A PCB named "jahn" does not exist
>
```

pcb all:

This command displays all PCBs (process control block) stored within the system(Ready, Blocked, Suspended, and Suspended Blocked processes). The value in parentheses after the names of the queues indicate the size of the queues. The number in parentheses after each PCB name indicates the priority of the PCB.

```
> pcb all
Ready Queue (2):
brian (5)
john (4)

Suspended Ready Queue (1):
jay (7)

Blocked Queue (1):
jason (3)

Suspended Blocked Queue (0):
>
```

pcb ready:

This command displays all ready processes within the system(ready and suspended ready processes). The value in parentheses after *Ready Queue* and *Suspended Ready Queue* indicate the size of the queues. The number in parentheses after each PCB name indicates the priority of the PCB.

```
Suspended Blocked Queue (0):
> pcb ready
Ready Queue (2):
brian (5)
john (4)

Suspended Ready Queue (1):
jay (7)
>
```

pcb blocked:

This command displays all blocked processes within the system(blocked and suspended blocked processes). The value in parentheses after *Blocked Queue* and *Suspended Blocked Queue* indicate the size of the queues. The number in parentheses after each PCB name indicates the priority of the PCB.

```
> pcb blocked
Blocked Queue (1):
jason (3)

Suspended Blocked Queue (0):
>
```

R3 - Dispatching

yield:

This command yields the CPU to allow other processes to run. Hence command will yield to any other processes. If there are processes in the ready queue, they will be executed when the yield command is called.

```
Ready Queue (1):  
proc5 (5)  
  
Suspended Ready Queue (4):  
proc4 (4)  
proc3 (3)  
proc2 (2)  
proc1 (1)  
  
Blocked Queue (0):  
  
Suspended Blocked Queue (0):  
> yield  
proc5 dispatched  
proc5 dispatched  
proc5 dispatched  
proc5 dispatched  
proc5 dispatched  
>
```

loadr3:

This command will load all the processes defined in the proc3.c file into memory into a suspended ready state at any priority that the user chooses.

```

> loadr3
> pcb all
Ready Queue (0):

Suspended Ready Queue (5):
proc5 (5)
proc4 (4)
proc3 (3)
proc2 (2)
proc1 (1)

Blocked Queue (0):

Suspended Blocked Queue (0):
>

```

pcb resumeall :

This command unsuspends all PCBs. If the user correctly specifies a suspended process, they will notice it would have been moved to the ready queue, if it was originally in the suspended ready queue or the blocked queue if it was originally in the suspended blocked queue after running the command.

```

Suspended Ready Queue (4):
proc4 (4)
proc3 (3)
proc2 (2)
proc1 (1)

Blocked Queue (0):

Suspended Blocked Queue (0):
> pcb resumeall
Successfully resumed "proc4"

Successfully resumed "proc3"

Successfully resumed "proc2"

Successfully resumed "proc1"

> pcb all
Ready Queue (4):
proc4 (4)
proc3 (3)
proc2 (2)
proc1 (1)

Suspended Ready Queue (0):

Blocked Queue (0):

```

R4 - Dispatching

alarm [hour]: [minute]: [second] [message] :

This command creates an alarm for the specified [hour]: [minute]: [second]. The user must specify a [message] after the [second] parameter that will be displayed when the alarm is triggered. The alarm triggers during or after the time specified in the parameters. User must specify time using the correct format, [hh]:[mm]:[ss], or [h]:[m]:[s](if time values are single digit).

```
> alarm 11:57:00 hellop9
Alarm successfully added with message 'hellop' set for 9 seconds from now.
> time
11:57:06

!!!! Alarm triggered: 'hellop' !!!!

Current time:
11:57:06
>
```

R5 - Memory Management

mem showallocated:

This command displays a list of the currently allocated memory blocks to the user. The list is ordered by address with the lowest address coming first. There are two additional fields for the size of the allocated memory and type of the memory block.

```
> mem showallocated
{ Type=Allocated, Address=1114265, Size=12 }
{ Type=Allocated, Address=1114329, Size=12 }
{ Type=Allocated, Address=1114393, Size=12 }
{ Type=Allocated, Address=1114457, Size=12 }
{ Type=Allocated, Address=1114521, Size=1072 }
{ Type=Allocated, Address=1115645, Size=1072 }
```

mem showfree:

This command displays a list of the currently freed memory blocks to the user. The fields output are the same as in mem showallocated. This list is also ordered with the lowest addresses coming first.

```
> mem showfree
{ Type=Free, Address=1116769, Size=47496 }
```


R6 - Interrupt Driven I/O

Unfortunately, our group was not able to complete a working implementation of the interrupt driven I/O module. Instead, we have a few different branches of source code to demonstrate our progress:

The **master** branch currently contains a demo program that simply demonstrates that the device driver and interrupt handlers function as intended.

The **R6Integrated** branch contains our attempt at getting the interrupt driven I/O to work with the rest of our program. It works a little bit, but not well enough to warrant placing the work on master.

The **R5** branch contains the most recent fully operational version of our MPX project.