100505349 Systems Programming Usage Document

*100505349*

Contents

[Structure 2](#_Toc90471290)

[Usage 2](#_Toc90471291)

[Basic commands 2](#_Toc90471292)

[Operating Modes 3](#_Toc90471293)

[Graphics Commands 3](#_Toc90471294)

[Capacity 4](#_Toc90471295)

[Presentation 5](#_Toc90471296)

[Locking 6](#_Toc90471297)

[Limitations and Features 6](#_Toc90471298)

# Structure

Made from the xv6 groundwork, this operating system implements the ability to generate and display simple graphics on a 320x200 window. The program that allows interaction with these graphics, Painter, is optimised to make use of both device contexts, locking and action queueing – it therefore can be run across multiple CPUs without having data conflict, and can process complex images based on the inputs provided by a user.

Painter can be accessed by writing the command word “Painter” into the console upon start-up – this process enters the scope of the Painter program after which all commands entered are interpreted by the painter program. It should be noted that for this program to work as intended, the OS must include the following unmodified files (as well as any other dependencies associated with the xv6 operating system):

* Painter.c
* Graphics.c
* GraphicsQueueables.c
* hdc.h
* rect.h

These files each contain necessary components of the Painter program, either through direct reference or access using system calls or similar functions. When examining the implementation of functions for this system, these files contain the appended code that constitutes the graphics system.

# Usage

When entering the context of the painter function, the user will be prompted to enter a command into the console, denoted by the prefix ‘HDC>’ applied to the command line. When entering a command, it is important to note that all commands must start with the key word “Painter”, followed by a space and then any arguments relevant to the function. It should also be noted that the system accepts a limited number of characters as an input, with inputs over 50 characters in length producing potentially anomalous results. Additionally, all inputs accepted by the Painter program are case-sensitive, therefore accepted commands should be checked for case before submission.

## Basic commands

When entering the Painter user program, the user has access to the following commands:

*Painter -nHDC*

This command creates a new device context, the numeric value of which will be shown in the prefix to all following inputs. This program accepts a maximum of 5 simultaneous device contexts and attempting to access any more than this value will result in the program preventing the user from spawning a new device context. **This command must be input before any of the graphics-oriented commands can be used.**

*Painter -exit*

The exit command allows a user to exit the scope of the “Painter” program, clearing the device context alongside it. This should only be entered when a user is done with the graphics functions, as any data – such as that in the queue – will be dropped.

*Painter -help*

This prints a list of all commands available to user within their current processing mode, though ignores the existence of any device context when listing commands. The returned command list includes the arguments accepted by each command as well as a brief description of what the commands are capable of doing.

*Painter -toggleMode*

This system allows toggling between the two modes of the Painter program – queueing mode and system call mode. After entering this command, the list of available commands will differ as some commands become irrelevant to the mode in use.

## Operating Modes

As mentioned prior, the Painter system makes use of two operating modes – queueing and system call modes – which are stored in the space of the user program rather than the device context. While in the context of queueing mode, which is the default option when entering the Painter program, all processed graphics commands will be appended to a list of commands, with a capacity of 50 actions. After entering all the commands, the user must enter the “Painter -exec” command which will process all commands in the sequence they were input and then display the results to the user. It should be noted that when a user enters a command, the action is permanently appended to the queue, and cannot be removed by any means aside from executing the queue – for this reason, when creating a complex graphics queue all inputs should be carefully monitored before submission.

The other mode available to the system, system call mode, allows the direct processing of a single command at a time. After each input, the graphics process in question will be instantly interpreted and the user will be presented with the graphical representation of their command. Upon completing a command, the graphics buffer will be cleared - thereby clearing any previous inputs from the screen. This restriction does not apply to the graphics cursor movement (-m) or the pen declaration or acquisition commands (-dp and -gp) – though notably due to the non-queued nature of this mode, the latter two commands may incur unexpected results when used in tandem with another CPU, for example if a pen is redeclared by another CPU between getting the pen and using it the colour will represent the newly defined pen. Finally, it should be noted that some interactions, such as the previously mentioned movement or pen commands, which apply changes to the device context itself, may carry these changes if the mode is switched.

## Graphics Commands

When a hdc has been defined and the user has toggled to the desired setting, the following functions can be used, using square brackets to denote arguments to be supplied:

*Painter -p [X] [Y]*

The pixel drawing command accepts two numbers representing the pixel that the process will draw upon, without changing the movement position of the device context in any manner.

*Painter -l [X] [Y]*

The line drawing command draws a line to the specified coordinates from the current graphics cursor position. When this command is used, the graphics cursor position will be automatically set to the specified coordinates.

*Painter -m [X] [Y]*

The movement command allows the moving of the graphics cursor, which starts in the (0,0) position, without applying any graphics to the screen. When processed in queueing mode, this will be processed like any other graphics command would be (sequentially), whereas in system call mode the movement procedure will apply instantly without opening the graphics window, this property is also exhibited by the -dp and -gp commands.

*Painter -dp [INDEX] [R] [G] [B]*

The declare pointer command creates a new pen at the specified index with the specified colours. The colour values chosen are limited to the range 0 through 63, and it should be noted that declaring a pen applies between device contexts and therefore modifications to a pen will occur across the entire operating system. Finally, the use of this command will not immediately set the pen of the user to the specified index, rather requiring the -gp command to be used.

*Painter -gp [INDEX]*

The get pen command pulls the graphical data from a specified pen index, applying the colour to all subsequent graphics operations. This can be used to get the data from a newly defined pen or the previously mentioned system colours, but may also be used to select undefined data, the use of which may incur undefined effects.

*Painter -r [X0] [X1] [Y0] [Y1]*

The rectangle command draws a filled rectangle between the specified coordinates using repeated calls to the line drawing function. As the shape drawn is rectangular, only two coordinates are required, defining the bounds of the rectangle shape. When used, this procedure will not change the graphics cursor position in any form.

In addition to these commands, there are two mode specific commands that cannot be interpreted when the procedure is not in the appropriate processing mode. These are as follows:

*Painter -exec* **(Queueing specific)**

Executes the processing of the queued actions, locking the system until this interaction is completed, after which it will display the image to the user. Once done, this command will dispose of the device context and its queue, requiring a call to the -nHDC command to be made to permit further graphics interactions.

*Painter -e* **(System call specific)**

This command, representing an “example”, draws a square with predefined coordinates to the screen, in the process modifying the movement position of the device context. This is only meant to demonstrate the functionality of the system and its ability to draw multiple structures simultaneously without queueing.

## Capacity

Some arguments and system contracts have capacities or limits, this comes in two forms: value limitations and memory limitations. In the former, some arguments may be restricted to a certain range to reduce errors and stop accessing of memory outside the scope of the program. One prime example of a property limited by the system is the coordinates, which automatically fit any entered commands within the range of the graphics space (specifically, x must fall between 0 and 319 and y must fall between 0 and 199) – any inputs that exceed or precede these values will be set to the closest valid number. The latter limitation refers to limits applied to array-type constructs in the system, such as the limit of 5 simultaneous active device contexts or the limit of 50 input characters. While these limitations are in some cases not communicated to the user within the program itself, they are important to consider when planning how to utilise the system (particularly in the case of the queue, which has a limit of 50 queued commands and 20 arguments within each command.

In addition to this, each graphics command will check to ensure that a number of arguments supplied is equal to or over the expected amount and will only make use of the arguments in the position it expects. For example, if a user enters “Painter -p 10 20 30”, the system will draw to the position 10,20 while disregarding the trailing arguments. While exceeding the number of accepted arguments is not recommended, when done it will be automatically ignored by the command interpreter.

# Presentation

The produced code for this process follows a set of formatting rules to make the processing components legible and consistent, with the intention of allowing future expansion or code review by third parties. In general, the code follows the following:

* All variables lead with a lowercase letter, followed by an uppercase letter if the name enters a new “word”
* All functions lead with an uppercase letter, using the same uppercase syntax as variables when entering a new “word” in the title
  + This formatting rule is ignored in the case of functions defined by the specification, such as the system calls, which use the format defined in the task
* Comments are used throughout, particularly in complex areas, and are appended on the same line as the code they are referencing, except in some cases where the use of a new line is appropriate (such as when referring to commented out code or when the comment is too large for a single line)
* Graphics code and most interactions with the supplied arguments are split across the “graphics.c” file and “graphicsQueueables.c” file, the latter of which is included in the former. The queueables file mostly contains the source code for action queueing, whereas the graphics.c file mostly contains system calls. Any constructs required by both files are included in the queueables file
* The “Painter.c” file follows a strict formatting procedure, with each command starting with the word “Painter”, followed by the command denoted by a single dash line. Additionally, the Painter file serves merely as a director to the graphics systems and does not interact with the writing to graphics memory itself.

# Locking

Locking is used in the beginpaint and endpaint system calls only, as all other actions refer to memory that should be device context specific or has specific reasons for not implementing locking. There is only one spinlock used in the graphics program, which exists in the scope of the “graphics.c” file, as this is the location of the beginpaint and endpaint system calls.

The beginpaint call uses the lock to find the next available hdc object and write to it if applicable, this is done to ensure that two separate CPU “threads” do not access the same data simultaneously. When the hdc is written to the lock is released and is not re-acquired until the endpaint system call is made.

In the endpaint system call, the locking encapsulates the entirety of the queue processing (which is skipped if the painter is in system call mode) and the removal of the device context from memory. Locking is used in the queue processing procedure to ensure that only one CPU can write to the graphics buffer at a time – as well as largely preventing pen issues. When a pen is declared, the data written applies outside of the hdc in use, therefore it is possible that a pen may be changed by a different CPU. To mitigate this issue in queueing mode (which is at higher risk of this problem impacting the resulting graphics), the declare pen code is added to the queue and therefore encapsulated within the lock used when processing queued actions, thereby stopping another queue-mode thread from overwriting pen data until the first thread has completed its actions.

# Limitations and Features

The system as presented includes the requirements specified from stages 1 through 4, as by my personal understanding of the tasks, and includes extra features that demonstrate the potential of the developed system – such as the inclusion of the input loop within the scope of the user program.

Some issues do exist either as a product of restrictions made by the emulator or to preserve legacy code. During the course of this assignment, the emulator has demonstrated some issues that do not interact well with the processing requirements of the system – for example, the switching to graphics mode from the text window tends to pause on a random frame, requiring a user to interact with the window to proceed. This issue has been known to stop the user from viewing the result of the graphics calls, as the getch() command used to wait for an input interprets the interaction with the OS window as a call to resume text mode, returning the user to text mode before the user can view the image. Additionally, the emulator occasionally inputs data of its own accord into the console, which in some cases may be interpreted by the Painter program, in this case it is best to remove the junk data if possible or the input may produce unexpected results.

In terms of the mentioned “legacy code”, the decision to implement both queueing and system call mode was done very late into development, after which there was no time to make modifications to old code and test the system properly. The system call mode exists solely as a demonstration of tasks before stage 3 and 4, and in a publicly released format of the system, the system call mode would not exist. As a result of this, interactions between queueing mode and system call mode are not supported and may produce anomalous results, including issues with pen declaration and getting of pens (which has locking included in the queue functionality as previously mentioned, and therefore the system call mode has no lock), as well as depreciated capacities or restrictions such as the checking of hdcs (a process now handled by the user program) or the restriction on overwriting certain pen values present in the system call pen declaration but not in the queueing mode. In conclusion, the queueing mode should be interpreted as the “product” of the assignment, whereas the system call mode functions exist as a demonstration of certain aspects of the program,