

Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
кафедра Информатики

Дисциплина: Архитектура вычислительных систем (АВС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту на тему

TASK МЕНЕДЖЕР ДЛЯ MACOS

Студент: гр.753505 Горбачёнок К.Н.

Руководитель: ассистент кафедры информатики

Леченко А.В.

Минск 2019

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ	2
ВВЕДЕНИЕ	3
1. Бэкенд часть приложения	4
1.1 Жизнь без /proc	4
1.2 XPC Service	5
1.3 Команда top	6
1.4 Команда ps	7
1.5 Убивание процессов	8
2. Интерфейс приложения.....	9
ЗАКЛЮЧЕНИЕ.....	11
СПИСОК ИСТОЧНИКОВ	12

ВВЕДЕНИЕ

Таск менеджер или по-другому диспетчер задач - это программа позволяющая смотреть статистику по используемым ресурсам, таким как память и ЦПУ, процессами системы, а так же, осуществляющая доступ к управлению состоянием этих процессов. Программа предоставляет пользователю информацию о системе в двух вариантах: графическом и текстовом.

Целью данной курсовой работы было создание такого приложения с использованием нативных инструментов разработки под систему macOS, а именно написать проект полностью на современном Swift, как для бэкенд части приложения, так и для написания интерфейса используя, для последнего, новый декларативный фреймворк SwiftUI, представленный этим летом на WWDC, и позволяющего реализовывать интерфейс стандартизированной формой на всех системах.

Для получения информации о процессах рассматривалось много вариантов, начиная от псевдо-файловой системы /proc, которой, к сожалению, не оказалось на macOS, и заканчивая низкоуровневыми “сишными” библиотеками, такими как libproc и sysctl, но из-за скудной документации и отсутствия гарантий работы на новых системах (последнее обновление документации датируется 2010м годом), было принято решение использовать стандартные unix-команды, такие как top и ps.

1. Бэкенд часть приложения

1.1 Жизнь без /proc

Изначально мною планировалось получать информацию о процессах системы через псевдо-файловую систему /proc, известную всем активным пользователям Linux-систем, которая предоставляет интерфейс к структурам ядра системы. К сожалению, хоть Linux и macOS и UNIX-системы, базируются они на разных системах. В случае macOS - это BSD, в которой нет реализации /proc.

Далее я продолжил свое исследование, чтобы узнать, как все таки получить информацию о процессах. После изучения огромного количества статей и форумов, у меня осталось несколько вариантов: библиотека libproc, sysctl и парсить результаты команд ps и top. В результате я остановился на последнем и сейчас поясню почему.

Данные библиотеки, хоть и предоставляют информацию о процессах системы не имеют вменяемой документации (несколько строк комментариев к функциям в хедере), для того, чтобы разобраться в них самому и потом встроить работу с ними в свой проект. Еще больше я убедился в этом, после прочтение форума, где человек задавался точно таким же вопросом как и я и на что ему официальный представитель Apple заявил, что все эти библиотеки, являются внутренним API для разработчиков компании, поэтому они не могут гарантировать совместимости с новыми версиями системы и лучшим выходом будет использовать команды терминала, что я в итоге и сделал.

Еще одной причиной такого выбора послужило то, что непонятно было можно ли вообще получить информацию по какому-то процессу по его PID'у (Process ID), хотя стоит отметить, что статическую информацию, такую как количество ядер процессора, название, общее использование, получить можно было, но хотелось единства написанного кода, поэтому было принято решение все таки использовать ps и top.

1.2 XPC Service

Для того, чтобы использовать команды `ps` и `top` в проекте необходимо было создать отдельный сервис именуемый XPC.

XPC Service - это некоторый сервис, который выполняет работу не требующую активного взаимодействия пользователя, которая выполняется в фоне, а родительское приложение, запустившее и использующее этот сервис, взаимодействует с ним и получает результаты через абстракции реализованные через интерфейсы. Общая схема взаимодействия изображена на рис. 1

Figure 4-1 The NSXPC architecture

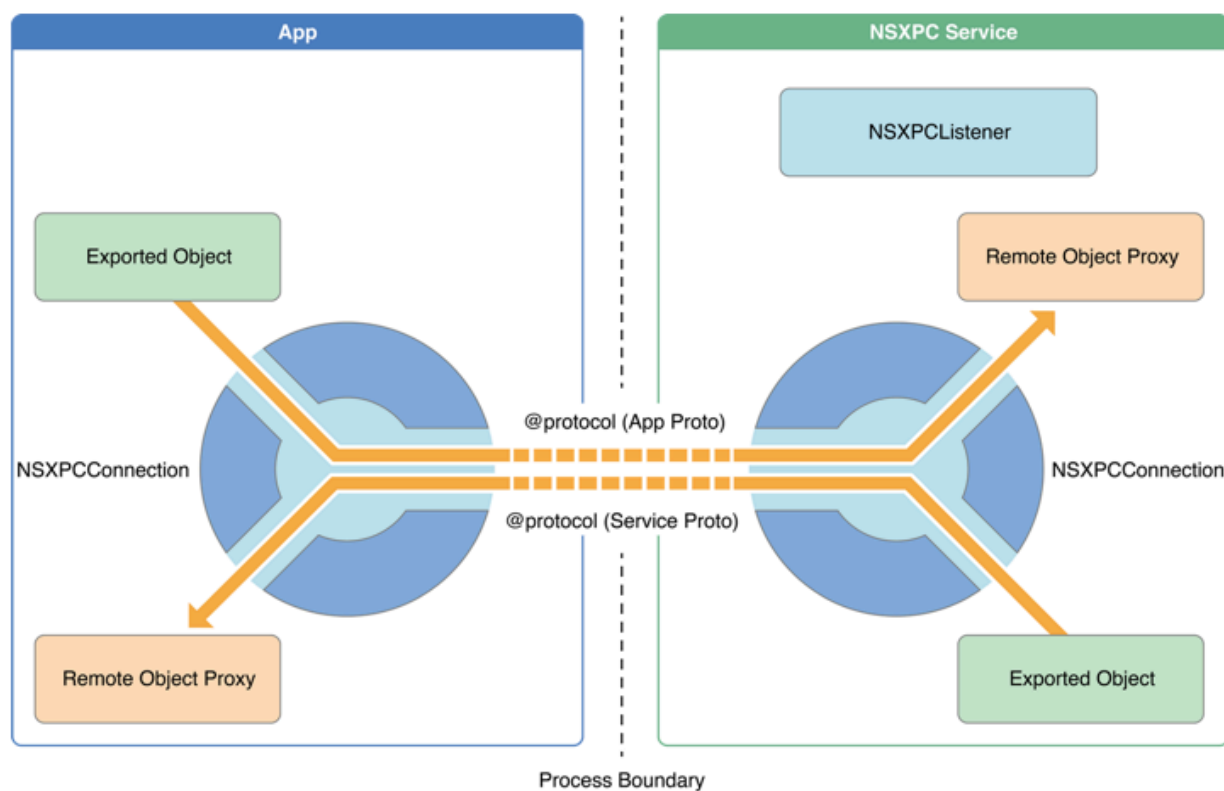


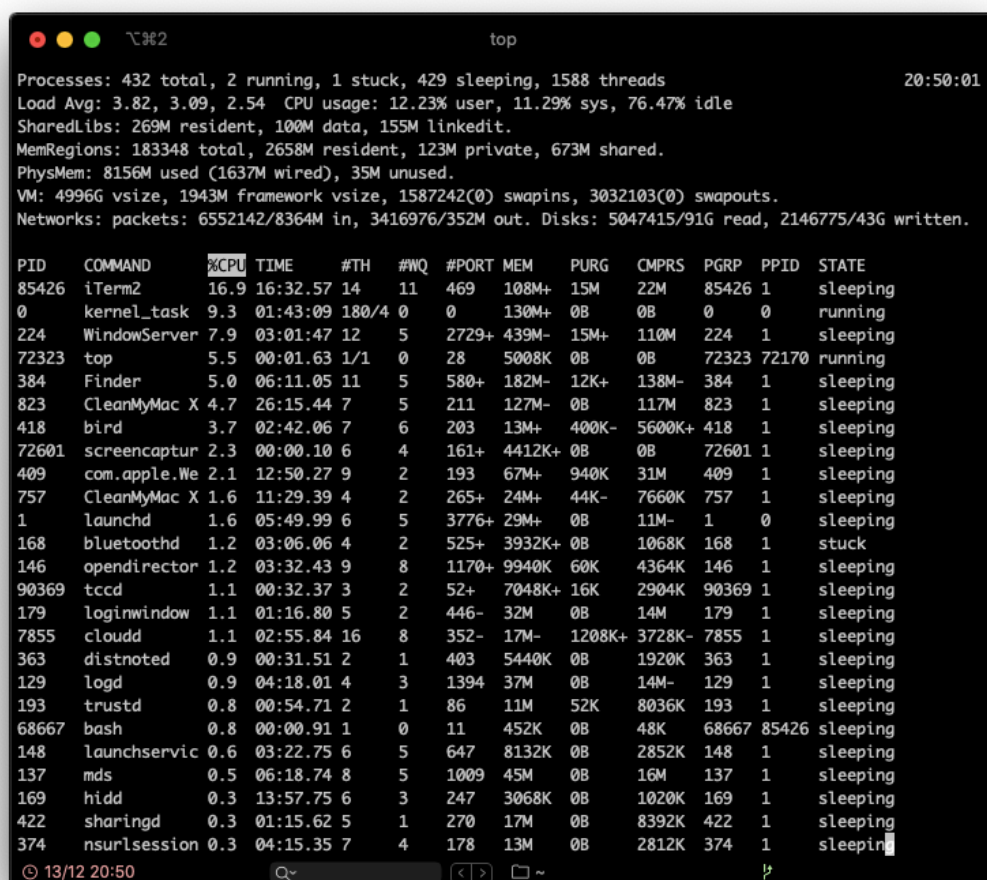
Рис.1. Схема взаимодействия

Такой сервис необходим из-за некоторых ограничений, связанных с повышенной безопасностью предоставляемой macOS её пользователям. Такая безопасность достигается за счет так называемых **App Sandbox** - технология контроля доступа, применяемая на уровне ядра системы. В своей сути эта технология ограничивает приложение, так, что оно имеет доступ только к

собственным данным и некоторым данным пользователя, которые не могут быть повреждены злоумышленниками. Поэтому в случае, если например ваше приложение запросит доступ к микрофону компьютера, оно сразу получит отказ от системы. Для этого нам и нужен XPC Service, так как он не будет ограничен Sandbox родительского приложения, будет выполняться как daemon-процесс, и сможет получать доступ, например на выполнение команд ps и top, при этом оставаясь частью приложения.

1.3 Команда top

Команда top в данном приложении используется для получения общей информации о загрузке системы. Эта команда выводит общую информацию о загрузке системы, а так же все процессы выполняемые на компьютере (рис. 2).



```
Processes: 432 total, 2 running, 1 stuck, 429 sleeping, 1588 threads
Load Avg: 3.82, 3.09, 2.54 CPU usage: 12.23% user, 11.29% sys, 76.47% idle
SharedLibs: 269M resident, 100M data, 155M linkedit.
MemRegions: 183348 total, 2658M resident, 123M private, 673M shared.
PhysMem: 8156M used (1637M wired), 35M unused.
VM: 4996G vsize, 1943M framework vsize, 1587242(0) swapins, 3032103(0) swapouts.
Networks: packets: 6552142/8364M in, 3416976/352M out. Disks: 5047415/91G read, 2146775/43G written.

PID    COMMAND    %CPU    TIME    #TH    #INQ    #PORT    MEM    PURG    CMPRS    PGRPR    PPID    STATE
85426   iTerm2      16.9    16:32.57 14     11     469     108M+  15M    22M     85426  1     sleeping
0       kernel_task 9.3     01:43:09 180/4  0       0       130M+  0B      0B      0       0       running
224     WindowServer 7.9     03:01:47 12     5       2729+   439M-  15M+   110M    224    1     sleeping
72323   top         5.5     00:01.63 1/1    0       28      5008K  0B      0B      72323  72170 running
384     Finder      5.0     06:11.05 11     5       580+    182M-  12K+   138M-   384    1     sleeping
823     CleanMyMac X 4.7     26:15.44 7       5       211     127M-  0B      117M    823    1     sleeping
418     bird        3.7     02:42.06 7       6       203     13M+   400K-   5600K+  418    1     sleeping
72601   screencaptur 2.3     00:00.10 6       4       161+    4412K+ 0B      0B      72601  1     sleeping
409     com.apple.We 2.1     12:50.27 9       2       193     67M+   940K    31M     409    1     sleeping
757     CleanMyMac X 1.6     11:29.39 4       2       265+    24M+   44K-    7660K   757    1     sleeping
1       launchd     1.6     05:49.99 6       5       3776+   29M+   0B      11M-    1       0     sleeping
168     bluetoothd  1.2     03:06.06 4       2       525+    3932K+ 0B      1068K   168    1     stuck
146     opendirector 1.2     03:32.43 9       8       1170+   9940K  60K     4364K   146    1     sleeping
90369   tcdd        1.1     00:32.37 3       2       52+     7048K+ 16K     2904K   90369  1     sleeping
179     loginwindow 1.1     01:16.80 5       2       446-    32M     0B      14M     179    1     sleeping
7855   cloudd      1.1     02:55.84 16      8       352-    17M-   1208K+  3728K-  7855  1     sleeping
363     distnoted   0.9     00:31.51 2       1       403     5440K  0B      1920K   363    1     sleeping
129     logd        0.9     04:18.01 4       3       1394    37M     0B      14M-    129    1     sleeping
193     trustd      0.8     00:54.71 2       1       86      11M     52K     8036K   193    1     sleeping
68667   bash        0.8     00:00.91 1       0       11      452K    0B      48K     68667  85426 sleeping
148     launchservic 0.6     03:22.75 6       5       647     8132K  0B      2852K   148    1     sleeping
137     mds         0.5     06:18.74 8       5       1009    45M     0B      16M     137    1     sleeping
169     hidd        0.3     13:57.75 6       3       247     3068K  0B      1020K   169    1     sleeping
422     sharingd    0.3     01:15.62 5       1       270     17M     0B      8392K   422    1     sleeping
374     nsurlsession 0.3     04:15.35 7       4       178     13M     0B      2812K   374    1     sleeping
```

Рис.2. top

Общий вид команды `top` используемой в программе получился вот такой:

```
top -l 1
```

Где `-l` - означает использовать `sample-mode` выполнения команды, `1` - количество итераций команды.

В отличие от Linux, в macOS нельзя сделать запрос на одну итерацию команды `top`, так чтобы она выводила правильный процент потребления CPU для каждого процесса, вместо нее используется команда `ps`, о которой далее.

1.4 Команда `ps`

Команда `ps` используется для создания `snapshot` текущих процессов системы. Она является основной в программе, так как используется практически для всего: выводит процессы и всю информацию о них. В свою очередь эта информация используется для построения графиков использования CPU и памяти.

Мною используются следующие данные из этой команды:

- PID - id процесса;
- Name - Имя процесса (благодаря параметру `-c` в команде запроса выводится имя вместо полного пути);
- CPU - процент времени CPU используемого процессом;
- MEM - процент использования памяти процессом;
- CPU Time - время работы процесса (исключает ожидание I/O);
- User - пользователей в пространстве которого исполняется этот процесс;
- State - состояние процесса.

Выделяют следующие возможные состояния процесса:

- D - непрерывный сон (обычно IO);
- I - Idle thread ядра;
- R - выполняется или выполняем (в очереди выполнения);
- S - прерываемый сон (ожидает выполнения события);
- T - остановлен контрольным сигналом;

- t - остановлен дебаггером во время отладки;
- W - пагинация (устарело с 2.6.xx ядер);
- X - мертв (не должен быть виден);
- Z - мертвый ("зомби") процесс, завершен и ожидает считывания кода завершения процесса родительским процессом;
- < - высокий приоритет;
- N - низкий приоритет;
- L - страницы заблокированы в памяти (для реального времени и пользовательских операций ввода-вывода);
- s - лидер сессии ;
- l - многопоточный (использует CLONE_THREAD как NPTL pthreads);
- + - находится в группе первостепенного выполнения.

Состояния могут комбинироваться.

Общий вид команды:

```
ps aux -c
```

Где aux - выводит полный список всех процессов от всех пользователей,
-c - выводит нормально название процесса, вместо его пути в системе.

1.5 Убивание процессов

Осуществляется стандартной командой kill имеющей следующий вид:

```
kill -9 [pid]
```

Где -9 - немедленное прекращение процесса (может привести к повреждению несохраненных данных), [pid] - ID процесса, который нужно убить.

2. Интерфейс приложения

Интерфейс приложения выглядит следующим образом рис. 3.

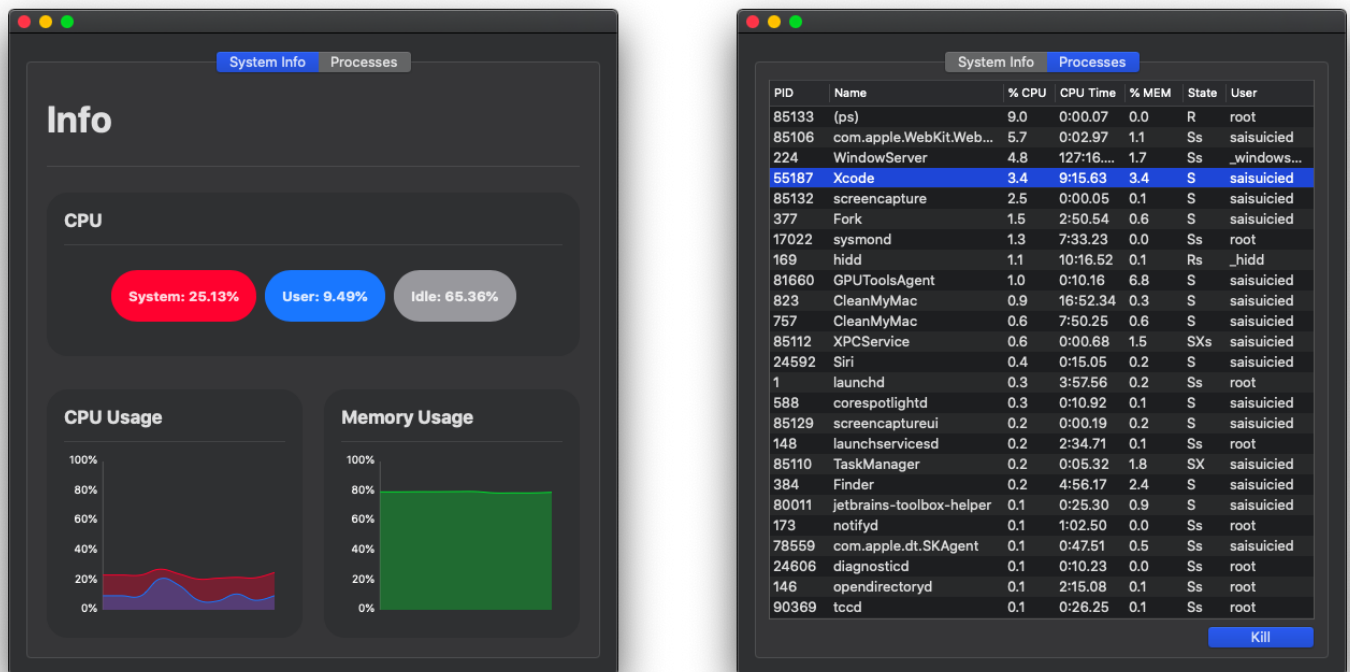


Рис.4. Интерфейс приложения

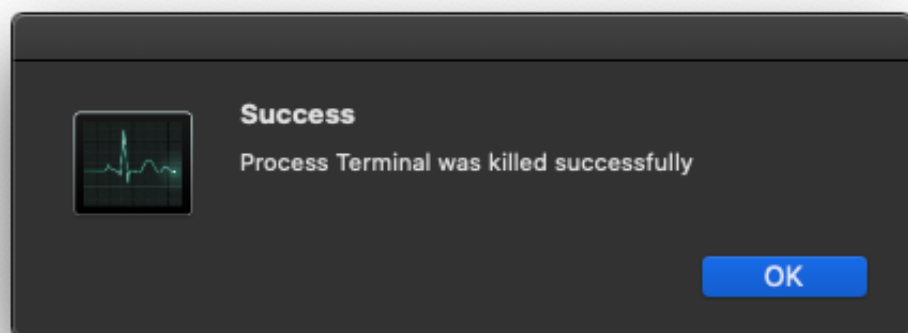


Рис.5. Pop-up

Интерфейс реализует полностью все возможности приложения:

- Просмотр общей информации о системе;

- Графическое изображение потребления CPU текущим пользователем и системой;
- Графическое изображение используемой оперативной памяти;
- Список всех процессов в системе;
- Возможность убить процесс;
- Убивание происходит при нажатии на кнопку Kill, либо по нажатию на клавишу Enter;
- По нажатию, в зависимости от того, успешно произошло убийство процесса или нет, появляется поп-ап с результатом (рис. 5).

ЗАКЛЮЧЕНИЕ

В результате выполнения проекта получилось нативное приложение для macOS, полностью покрывающее поставленную задачу. Оно предоставляет пользователю информацию о ресурсах системы, а так же наглядно иллюстрирует эту информацию, позволяет просматривать процессы, управлять ими (убивать). Приложение соответствует стандартам дизайна платформы (Apple Human Interface Guidelines) и использует современный фреймворк для создания этого интерфейса.

В ходе выполнения работы я получил более углубленные знания о работе системы macOS, о том как выполняются процессы, как они исполняются и в каком состоянии они могут находиться, чем она отличается от Linux и т.п.

Так же я получил знания о новом подходе в написании нативных приложений для macOS, чем оно отличается от мобильной разработки и многое другое.

СПИСОК ИСТОЧНИКОВ

1. https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/BPSystemStartup/Chapters/CreatingXPCHServices.html#//apple_ref/doc/uid/10000172i-SW6-SW1
2. <https://developer.apple.com/library/archive/documentation/Security/Conceptual/AppSandboxDesignGuide/AboutAppSandbox/AboutAppSandbox.html>
3. <http://osxbook.com/book/bonus/ancient/procfs/>
4. <https://opensource.apple.com/source/xnu/xnu-2422.1.72/libsyscall/wrappers/libproc/libproc.h.auto.html>
5. <https://opensource.apple.com/source/xnu/xnu-201.5/bsd/sys/sysctl.h.auto.html>
6. <https://ru.wikipedia.org/wiki/MacOS>
7. <http://man7.org/linux/man-pages/man5/proc.5.html>
8. <https://ss64.com/osx/top.html>
9. <http://man7.org/linux/man-pages/man1/ps.1.html>
10. <https://developer.apple.com/design/human-interface-guidelines/>
11. <https://developer.apple.com/documentation/swiftui>