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

Таск-менеджер для android

Студент: гр.753504 Кузнечик В.А.

Руководитель: ассистент кафедры информатики

Леченко А.В.

Минск 2019

Содержание

Введение

Целью данного проекта было создание таск-менеджера под операционную систему android. В функциональность таск-менеджера входит отображение текущей загруженности системы, процессов, которые сейчас запущены, и статистику по ним, такую как занимаемая память, потребление времени процессора, состояние и название. Также процессами можно управлять, в данном приложении их можно убивать.

Поскольку android работает на ядре Linux, то получение информации о системе можно получать из виртуальной файловой /proc или выполнением определённых команд (например, top и ps с разными опциями).

1. Виртуальная файловая система /proc

Для получения информации о загруженности процессора я использовала псевдофайловую систему **/proc**.

Файловая система /proc - это виртуальная файловая система в операционной системе Linux, и файлы в этом каталоге не занимают места на жестком диске. Поскольку android использует ядро Linux, то /proc здесь также доступна. Файловая система /proc это интерфейс ко внутренним структурам данных в ядре, что можно использовать для получения информации о системе и изменения некоторые параметров ядра во время работы системы.

Многие командные оболочки собирают информацию из файлов в /proc, форматируют их и выводят результат пользователю (`top, ps` и другие). Также существует специальный подкаталог /proc/sys. Он позволяет вам просматривать параметры ядра и изменять их.

Чтение информации из этой виртуальной системы разрешено только до 25 api включительно.

Из директории proc/stat я брала первую строку с информацией по времени работы всего процесса. Первая строка агрегирует все нижние строки и выглядит в общем виде так:

```
CPU [user] [nice] [system] [idle] [iowait] [irq][softirq]
```

- user — время, затраченное на работу программ пользователей
- system — время, затраченное на работу процессов ядра
- nice — время, затраченное на работу программ с измененным приоритетом
- idle — простой процессора
- iowait — время, затраченное на завершение ввода-вывода
- irq — время, затраченное на обработку hardware-прерываний
- softirq — время, затраченное на работу обработку software-прерываний

Чтобы определить процентное значение загруженности процесса, надо посчитать отношения времени простоя (idle) на общее время работы процессора (сумма всех значений в строке) и вычесть это значение из единицы, умножить результат на 100%.

Изначально статистику по работающим процессам я получала из /proc/[pid]/stat. Я делала это следующим образом: считывала строку с информацией по процессу и конвертировала её в значения. Из файла получала pid процесса (process id), state, название процесса, количество использованной виртуальной памяти, а для подсчёта затраченной мощности процессора нужны были значения utime (user time) и systime (system time). Utime - время, затраченное на выполнение кода программы и библиотек, а sustime - время системных вызовов ядра от имени программы.

Поскольку значения показывают время работы процессора, начиная с uptime, требовалось зафиксировать значения utime и systime в определённый

момент времени, затем подождать примерно 500 мс и считать их ещё раз. Итоговое процентное отношения высчитывалось как

$$\frac{(utime_2 - utime_1) + (systime_2 - systime_1)}{cputime}$$

2. Команда top

Однако на android можно только получать процессы, относящиеся к данному приложению.

Но чтобы получать все процессы и убивать другие процессы на android устройстве, нужны root права, так можно исполнять команды от суперпользователя. Таким образом, я каждые пару секунд запускаю исполнение команды `su top -n 1`, которая выводит одну итерацию команды `top`:

```
val su = Runtime.getRuntime().exec("su")
val outputStream = DataOutputStream(su.outputStream)
outputStream.writeBytes("top -n 1" + "\n")
outputStream.flush()
val reader=BufferedReader(InputStreamReader(su.inputStream))
outputStream.writeBytes("exit" + "\n")
outputStream.flush()
outputStream.close()
su.destroy()
```

Здесь создаю процесс su, который выполняет команды от имени суперпользователя, затем в поток вывода записываю исполняемые команды и создаю объект reader, который считывает вывод команд. По завершению поток закрывается и процесс разрушается.

Так получаю процессы и информацию о них, они представляются следующим образом в виде списка

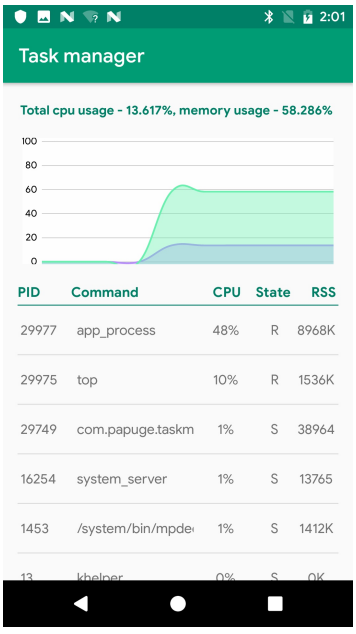

Рисунок 1, экран приложения

Здесь есть такие значения, как *state* и *rss*. State - состояние, в котором сейчас находится процесс. Различают следующие состояния:

- R : процесс выполняется в данный момент
- S : процесс ожидает (т.е. спит менее 20 секунд)
- I : процесс бездействует (т.е. спит больше 20 секунд)
- D : процесс ожидает ввода-вывода (или другого недолгого события), непрерываемый
- Z : zombie или defunct процесс, то есть завершившийся процесс, код возврата которого пока не считан родителем
- T : процесс остановлен
- W : процесс в свопе
- < - процесс с отрицательным значением nice
- N - процесс с положительным значением nice
- L : real-time процесс, имеются страницы, заблокированные в памяти.
- s : лидер сессии

Значение nice положительно, если приоритет исполнения процесса был *повышен* (повышать приоритет может только суперпользователь), а отрицательный nice соотвественно значит, что приоритет был *понижен* пользователем.

RSS - это размер резидентного раздела который используется, чтобы показать, сколько оперативной памяти выделено этому процессу. Он не включает память из раздела подкачки. Он включает в себя память из общих библиотек, пока страницы из этих библиотек фактически находятся в памяти. Он включает в себя всю память стека и кучи. Поскольку RSS хранит ещё размер используемых общих библиотек, то сложив все значения по процессам, мы можем получить значение больше фактического размера памяти.

3. Убийство процессов

Убивать процессы по значению pid можно командой `su kill -9 [pid]` или же вызовом метода `android.os.Process.killProcess(pid)`, однако это на android работает довольно странно, потому что там есть ограничения по завершению процессов. Но когда я убила один из процессов, телефон перезагрузился.

4. Мониторинг памяти

Общее процентное значения потребления памяти я получала из класса `ActivityManager`, который предоставляет информацию о процессах в системе. У объекта этого класса можно вызвать метод `getMemoryInfo()` и передать туда экземпляр класса `ActivityManager.MemoryInfo`. Затем у этого экземпляра можно получить свойства `availMem` (размер доступной памяти) и `totalMem` (общий размер памяти). Процентный расход памяти получаем по формуле

$$(1 - \frac{availMem}{totalMem}) * 100$$

4. Визуальная часть приложения

В приложении присутствует визуальное отображение занятости процессора и оперативной памяти в виде графиков (занятость памяти - фиолетовый цвет, а процессора - зелёный). Для построения графиков использовалась библиотека *MPAndroidChart* (https://github.com/PhilJay/MPAndroidChart). Данные обновляются каждый 3 секунды.

Для отображения списка процессов и обновления данных использовался компонент *RecyclerView.* Снимать задачи можно свайпом влево.

Заключение

В итоге выполнения курсового проекта было сделано приложение, которое отображает загруженность процессора и занятость оперативной памяти, иллюстрирует эти соотношения в изменяющихся графиках. Также пользователь видит список всех процессов в системе, может их убивать и смотреть по ним различную статистику.

В ходе проекта я узнала о многих способах получения информации о процессах в операционной системе android, а ещё столкнулась с рядом ограничений системы (сделано ради безопасности). Во-первых, чтение из /proc разрешено только до 25 api, притом получать pid можно только у процессов своего приложения. Чтобы получать информацию обо всех процессах, нужен root. И тогда на девайсе можно запускать команды вроде top, ps.

Я узнала базовые вещи о том, как получать статистику о загруженности процессора, памяти, значения при выводе команды `top` и как убивать процессы.

Литература

1. http://man7.org/linux/man-pages/man5/proc.5.html
2. https://linux.die.net/man/1/top
3. https://developer.android.com/reference/android/app/ActivityManager
4. https://rosettacode.org/wiki/Linux_CPU_utilization