تکلیف سری دوم سیستم عامل
سارا برادران ۹۶۲۴۱۹۳

سوال ۱

الف) تمام حالات ممکن برای اجرای زیر برنامه ها در زیر آمده است و با توجه به مقدار حافظه ای که در هر یک از اجراهای زیر مورد نیاز است می توان گفت ماکزیمم این میزان حافظه ، همان حداقل فضای مورد نیاز جهت اجرای برنامه اصلی به شیوه مدیریتی روی هم گذاری است. مثلا زمانی که زیر برنامه ی A در حال اجرا است زیر برنامه D نیز فراخوانی می شود در داخل زیر برنامه D نیز ابتدا زیربرنامه E فراخوانی خواهد شد و پس از اتمام آن زیر برنامه B فراخوانی خواهد شد پس در یک زمان واحد باید بتوان مقدار حافظه مورد نیاز جهت اجرای مجموعه زیربرنامه های A , D , E را تامین کرد و به همین ترتیب در یک زمان واحد نیز باید بتوان حافظه مورد نیاز برای اجرای مجموعه زیر برنامه های A , D , B را نیز فراهم کرد. برای سایر مجموعه ها نیز به همین ترتیب.

sub programs running at the same time ----------------------------------

A , D , E $\rightarrow 5+15+7=27$
A , D , B $\rightarrow 5+15+9=29\rightarrow$ at least memory capacity which is needed
A , F , C $\rightarrow 5+10+10=25$
B $\rightarrow 9$
C $\rightarrow 10$
D , E $\rightarrow 15+10=25$
D , B $\rightarrow 15+9=26$
E $\rightarrow 7$
F , C $\rightarrow 10+10=20$

ب)

مزایا : این روش مقدار حافظه مورد نیاز برای اجرای برنامه که می بایست از رم اشغال شود را کاهش می دهد به علاوه می تواند روی زمان اجرا برنامه نیز تاثیر داشته باشد.

معایب : در حقیقت این برنامه نویس است که می بایست روش روی هم گذاری و بخش های مجزای مورد نیاز از حافظه (overlay ها) را مدیریت کند. و سیستم عامل صرفا آن ها را جایگذاری می کند اما در تقسیم بندی نقشی ندارد. بنابراین برنامه نویس می بایست از جزئیات حافظه اطلاع داشته باشد که خود می تواند سختی هایی ایجاد نماید. به علاوه در این روش این دشواری می تواند وجود داشته باشد که ماژول هایی در برنامه که وابسته به هم هستند نتوانند در قالب ovelay های مجزا تقسیم بندی شوند. بنابراین پیاده سازی این روش روی هم گذاری برای مدیریت حافظه پیچیده و در مواردی حتی غیر ممکن است.

سوال ۲

الف) هر صفحه حاوی ۲۰۴۸ بایت می باشد و فضای آدرس دهی ۶۴ بیتی است پس در مجموع ۱۱ بیت برای offset نیاز داریم و تعداد کل صفحات برابر $\frac{2^{64}}{2^{11}} = 2^{53}$ می باشد. از آنجایی که هر سطح از page table در یک صفحه قرار می گیرد و می تواند حاوی ۲۰۴۸ بایت باشد و هر PTE طبق صورت سوال ۴ بایت است پس در یک سطح از جدول حداکثر $\frac{2048}{4} = 512$ PTE قرار می گیرد. پس ۹ بیت از فضای ادرس برای یک سطح مورد نیاز است به همین ترتیب می توان ۵ سطح در نظر گرفت که در هر سطح ۵۱۲ مورد PTE موجود است نهایتا ۸ بیت باقی خواهد ماند که می توان گفت سطح آخر دارای ۲۵۶ مورد PTE است. پس در مجموع ۶ سطح خواهیم داشت.

Table 1: address structure

8 bit pt 6	9 bit pt 5	9 bit pt 4	9 bit pt 3	9 bit pt 2	9 bit pt 1	11 bit offset

ب) در حالتی که هر سطح از جدول در ۲ صفحه قرار بگیرد در مجموع می توان گفت هر سطح دارای $1024 = \frac{2\times 2048}{4}$ مورد PTE است پس تعداد ۱۰ بیت از فضای آدرس برای هر سطح مورد نیاز است در مجموع می توان ۵ سطح دارای ۱۰۲۴ مورد PTE و یک سطح دارای ۸ مورد PTE در نظر گرفت پس مجددا ۶ سطح خواهیم داشت.

Table 2: address structure

3 bit pt 6	10 bit pt 5	10 bit pt 4	10 bit pt 3	10 bit pt 2	10 bit pt 1	11 bit offset

سوال ۳

LRU ⇐ در سیاست LRU صفحه ای چایگزین خواهد شد که آخرین زمان دسترسی به آن دورتر باشد در میان صفحات موجود در حافظه دورترین دسترسی مربوط به صفحه با شماره مجازی ۱ می باشد که در زمان ۲۳۹ مورد استفاده قرار گرفته است پس در این سیاست صفحه با شماره مجازی ۴ جایگزین صفحه با شماره مجازی ۱ خواهد شد.

CLOCK ⇐ در سیاست clock یکی از صفحاتی که reference bit مربوط به آن ها ۰ است به عنوان قربانی و صفحه چایگزین شونده انتخاب می شود با توجه به اینکه در جدول داده شده تنها صفحه با شماره مجازی ۲ دارای reference bit صفر می باشد لذا در این سیاست صفحه با شماره مجازی ۴ جایگزین صفحه با شماره مجازی ۲ خواهد شد.

سوال ۴ طبق صورت سوال هر صفحه حاوی ۲۵۶ بایت است به علاوه حافظه مورد نیاز برای ذخیره سازی کل عناصر آرایه برابر $128 \times 128 = 2^{14}$ بایت می باشد. بدین ترتیب می توان گفت تعداد صفحات لازم برای ذخیره این آرایه $\frac{2^{14}}{2^8} = 2^6$ می باشد.

الف) در این حالت می توان گفت اولین بار که به کد برنامه نیاز داریم یک page fault رخ داده و سپس کد مربوط به برنامه در صفحه اول جایگزین می شود مجددا زمانی که نیاز به دسترسی ب اولین عضو آرایه می باشد یک page fault دیگر رخ داده و سپس عناصر ۱ تا ۲۵۶ آرایه در صفحه دوم جایگزین می شوند لذا برای دسترسی به ۲۵۵ عضو بعدی page fault رخ نخواهد داد سپس زمانی که به عضو ۲۵۷ آرایه نیاز داریم مجددا یک page fault رخ داده و سپس عناصر ۲۵۷ تا ۵۱۲ در صفحه سوم جایگزین شده و ضمن دسترسی به ۲۵۵ عضو بعد از عنصر ۲۵۷ ام page fault ای رخ نخواهد داد سپس برای دسترسی به عضو ۵۱۳ آرایه مجددا page fault رخ داده و این بار عناصر ۵۱۳ تا ۷۶۸ آرایه در صفحه اول که حاوی کد برنامه بود جایگزین می شود. اما بلافاصله پس از این جایگزینی مجددا به کد نیاز داریم لذا یک page fault رخ داده و کد برنامه در صفحه دوم جایگزین می گردد. و این چرخه تا خاتمه برنامه ادامه خواهد داشت. در کل می توان گفت ۶۴ بار برای دسترسی به اعضای آرایه و ۲۲ بار برای دسترسی به حافظه کد page fault رخ خواهد داد. پس تعداد رخداد $\text{page fault} : 64 + 22 = 86$

ب) از آنجایی که هر عنصر آرایه تنها یک بار مورد دسترسی قرار میگیرد و صفحه مربوط به حافظه کد نیز جدا از دیتا می باشد لذا الگوریتم جایگزینی اهیمتی پیدا نمی کند. با توجه یه اینکه المان های آرایه در ۶۴ صفحه مجزا قرار دارند و در هر صفحه ۲۵۶ عضو از آرایه جای می گیرد می توان گفت دسترسی به اولین عضو هر صفحه منجر به رخ داد page fault می گردد اما برای دسترسی به ۲۵۵ عضو بعدی هیچگونه page fault ای رخ نخواهد داد. علاوه بر بخش ذکر شده که مربوط به حافظه دیتا است نخستین بار که به کد برنامه جهت اجرای آن نیاز داریم یک page fault رخ می دهد که در نتیجه آن یکبار کد برنامه در صفحه مربوط به حافظه کد load شده و تا زمان خاتمه برنامه در آن باقی خواهد ماند و با چیزی جایگزین نمی شود. پس می توان گفت تعداد رخداد $\text{page fault} : 64 + 1 = 65$

سوال ۵ اگر از TLB استفاده نکنیم برای ترجمه هر آدرس مجازی به آدرس فیزیکی نیاز است به page table موجود در حافظه اصلی مراجعه کنیم به علاوه برای دسترسی به محتوای هر صفحه نیز می بایست یکبار به حافظه اصلی مراجعه کنیم لذا برای دسترسی به آدرس و محتوای ۵ صفحه داده شده لازم است در مجموع ۱۰ بار به حافظه اصلی مراجعه شود پس زمان دسترسی در کل برابر $10 \times 50ns = 500ns$ خواهد بود. حال اگر از TLB استفاده کنیم برای یافتن آدرس فیزیکی صفحات با آدرس مجازی ۲ و ۱ و ۳ ابتدا به TLB مراجعه می کنیم اما از آنجایی که این صفحات هر کدام بار اول است که مورد دسترسی قرار می گیرند لذا TLB miss رخ داده و نیاز است به page table موجود در حافظه اصلی مراجعه کنیم اما برای دسترسی به آدرس فیزیکی دو صفحه اخر یعنی صفحات با آدرس مجازی ۱ و ۲ چون آدرس فیزیکی این صفحات در دسترسی های قبلی وارد TLB شده و در آن موجود هست پس دیگر نیازی به مراجعه به حافظه اصلی نمی باشد. اما برای دسترسی به محتویات هر ۵ صفحه همچنان نیاز است ۵ بار به حافظه اصلی مراجعه کنیم. به علاوه زمانی که TLB miss رخ می دهد پس از مراجعه به حافظه و یافتن آدرس های فیزیکی نظیر هر آدرس مجازی مجددا به TLB بازگشته و آدرس های بدست آمده را در آن ذخیره می کنیم پس در مجموع ۲ بار به TLB مراجعه خواهیم کرد. پس زمان دسترسی در کل برابر $480 = 50 \times 5 + (50 + 10 + 10) \times 3 + 10 \times 2$ خواهد بود.

نتیجه ⇐ میزان بهبود کارآیی برابر $\frac{500-480}{500} \times 10 = 4\%$ می باشد.

سوال ۶

الف) valgrind ⇐ ابزاری است که می توان به کمک آن باگ ها و اشکالات مدیریت حافظه برنامه ها را شناسایی کرد. تعدادی از فلگ های قابل استفاده ضمن اجرا با این ابزار و کاربرد آن ها در تصویر زیر ذکر شده است.

- `--leak-check=full`: "each individual leak will be shown in detail"
- `--show-leak-kinds=all`: Show all of "definite, indirect, possible, reachable" leak kinds in the "full" report.
- `--track-origins=yes`: Favor useful output over speed. This tracks the origins of uninitialized values, which could be very useful for memory errors. Consider turning off if Valgrind is unacceptably slow.
- `--verbose`: Can tell you about unusual behavior of your program. Repeat for more verbosity.
- `--log-file`: Write to a file. Useful when output exceeds terminal space.

gdb ⇐ gdb در حقیقت مخفف GNU debugger است. که با سینتکس زیر می توان در ترمینال و cmd از آن استفاده کرد. اگر بخواهیم از gdb استفاده کنیم می بایست برنامه را با فلگ g- کامپایل کنیم.

```
gdb [-help] [-nx] [-q] [-batch] [-cd=dir] [-f] [-b bps] [-tty=dev] [-s symfile] [-e prog] [-se prog]
[-c core] [-x cmds] [-d dir] [prog[core|procID]]
```

تعدادی از آپشن های اجرای این برنامه و کاربرد ان ها در زیر آمده است :

run [args] این دستور فایل اجرایی را با آرگومان های داده شده اجرا می کند.

quit or q این دستور برای خروج از gdb می باشد.

break به کمک این دستور می توانیم در نقاط مختلف برنامه break point ایجاد نماییم تا جریان اجرای برنامه ضمن رسیدن به آن ها متوقف شود در تصویر زیر جزئیات بیشتری از نحوه استفاده از این دستور برای گذاشتن break point آورده شده است.

```
b
break [function name]
break [file name]:[line number]
break [line number]
break *[address]
break ***any of the above arguments*** if [condition]
b ***any of the above arguments***
```

delete با استفاده از این دستور می توان break point ها و check point های گذاشته شده را حذف کرد. اگر این دستور بدون آرگومان اجرا شود تمام break point های موجود حذف خواهند شد در تصویر زیر جزئیات بیشتری از نحوه استفاده از این دستور برای حذف break point ها آورده شده است.

```
d
delete
delete [breakpoint number 1] [breakpoint number 2] ...
delete checkpoint [checkpoint number 1] [checkpoint number 2] ...
```

set args [arg1] [arg2] به کمک این دستور می توان آرگومان های ورودی برنامه را تنظیم کرده و لذا پس از می توان دستور run را بدون آرگومان اجرا کرد.

ب) تصویر زیر نتیجه اجرای برنامه نوشته شده با استفاده از malloc و با استفاده از ابزار valgrind می باشد. همانطور که مشخص است ۴۰۰ بایت نشت حافظه رخ داده است.(برنامه نوشته شده تحت عنوان 6-malloc.c ضمیمه شده است.)

```
→ Desktop valgrind ./a.out
==16931== Memcheck, a memory error detector
==16931== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==16931== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==16931== Command: ./a.out
==16931==
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
==16931==
==16931== HEAP SUMMARY:
==16931==     in use at exit: 400 bytes in 1 blocks
==16931==   total heap usage: 2 allocs, 1 frees, 1,424 bytes allocated
==16931==
==16931== LEAK SUMMARY:
==16931==    definitely lost: 400 bytes in 1 blocks
==16931==    indirectly lost: 0 bytes in 0 blocks
==16931==      possibly lost: 0 bytes in 0 blocks
==16931==    still reachable: 0 bytes in 0 blocks
==16931==         suppressed: 0 bytes in 0 blocks
==16931== Rerun with --leak-check=full to see details of leaked memory
==16931==
==16931== For counts of detected and suppressed errors, rerun with: -v
==16931== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
→ Desktop
```

با استفاده از فلگ leak-check=full– می توان اطلاعات دقیق تری در مورد آدرس نشت حافظه بدست آورد. نتیجه اجرای این دستور در تصویر زیر قابل مشاهده است.

```
→ Desktop valgrind --leak-check=full ./a.out
==17194== Memcheck, a memory error detector
==17194== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==17194== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==17194== Command: ./a.out
==17194==
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
==17194==
==17194== HEAP SUMMARY:
==17194==     in use at exit: 400 bytes in 1 blocks
==17194==   total heap usage: 2 allocs, 1 frees, 1,424 bytes allocated
==17194==
==17194== 400 bytes in 1 blocks are definitely lost in loss record 1 of 1
==17194==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==17194==    by 0x10872B: main (in /home/sara/Desktop/a.out)
==17194==
==17194== LEAK SUMMARY:
==17194==    definitely lost: 400 bytes in 1 blocks
==17194==    indirectly lost: 0 bytes in 0 blocks
==17194==      possibly lost: 0 bytes in 0 blocks
==17194==    still reachable: 0 bytes in 0 blocks
==17194==         suppressed: 0 bytes in 0 blocks
==17194==
==17194== For counts of detected and suppressed errors, rerun with: -v
==17194== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
→ Desktop
```

ج) تصویر زیر نتیجه اجرای برنامه نوشته شده با استفاده از mmap و با استفاده از ابزار valgrind می باشد. همانطور که مشخص است هیچ نشت حافظه ای رخ نداده است.(برنامه نوشته شده تحت عنوان 6-mmap.c ضمیمه شده است.)

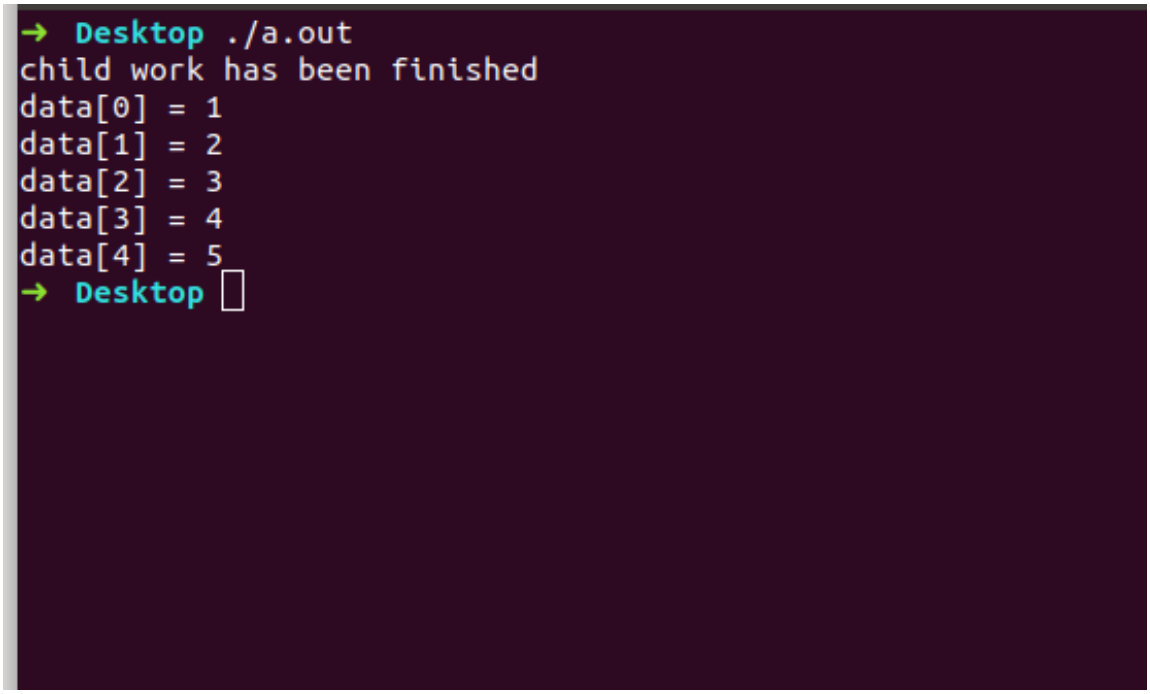
```
→ Desktop valgrind ./a.out
==17112== Memcheck, a memory error detector
==17112== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==17112== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==17112== Command: ./a.out
==17112==
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
==17112==
==17112== HEAP SUMMARY:
==17112==     in use at exit: 0 bytes in 0 blocks
==17112==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==17112==
==17112== All heap blocks were freed -- no leaks are possible
==17112==
==17112== For counts of detected and suppressed errors, rerun with: -v
==17112== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
→ Desktop
```

د) برنامه های نوشته شده تحت عنوان MapPrivate.c , MapShared.c ذخیره شده است.

MAP-SHARED : Share this mapping. Updates to the mapping are visible to other processes mapping the same region, and (in the case of file-backed mappings) are carried through to the underlying file.

MAP-PRIVATE : Create a private copy-on-write mapping. Updates to the mapping are not visible to other processes mapping the same file, and are not carried through to the underlying file.

تصویر زیر برنامه ای را نشان می دهد که در آن از فلگ MAP-PRIVATE استفاده شده است. همانطور که مشخص است نتیجه تغییر المان های آرایه توسط پروسس فرزند ، برای پروسس والد قابل مشاهده نیست.



تصویر زیر برنامه ای را نشان می دهد که در آن از فلگ MAP-SHARED استفاده شده است. همانطور که مشخص است نتیجه تغییر المان های آرایه توسط پروسس فرزند را در پروسس والد می توان مشاهده کرد.

سوال ۷

الف) در این برنامه آنچه در وهله اول به نظر می رسد این است مقدار متغیر pass در هیچ بلاکی برابر a یا همان ۹۷ نمی شود و در صورتی که عبارت "pass" به عنوان پسورد وارد شود correct password , successful login چاپ شده و در غیر اینصورت wrong password failed to login , چاپ می گردد. اما در حقیقت می توان گفت برنامه همواره این چنین عمل نمی کند از آنجایی که دستور gets() هیچ محدودیتی را بر روی سایز رشته ورودی اعمال نمی کند لذا در صورتی که یک عبارت با بیش از ۵ کاراکتر (با احتساب کاراکتر ۰ انتهای رشته) به عنوان پسورد ورودی وارد شود سرریز رخ خواهد داد و از آنجایی که تنها ۵ بایت از حافظه stack متعلق به buff می باشد و بایت های بعدی stack به متغیر pass اختصاص پیدا کرده است ، پسورد با طول بیش از ۵ کاراکتر(با احتساب کاراکتر ۰ انتهای رشته) داخل buff جا نمی شود و لذا سرریز آن در خانه های بعدی حافظه stack وارد شده و در فضای مربوط به متغیر pass ذخیره می گردد. لذا عملکرد برنامه به کلی تغییر خواهد کرد برای مثال اگر عبارت passaa به عنوان ورودی داده شود کاراکتر انتهایی a و نیز ۰ انتهای رشته سرریز شده در حافظه مختص متغیر pass ذخیره می گردد و لذا برنامه وارد بلاک if(pass == 'a') شده و عبارت successful login as admin چاپ خواهد شد اگرچه که ضمن مقایسه پسورد ورودی با عبارت "pass" عبارت wrong password چاپ می گردد زیرا مقدار passaa با pass برابر نمی باشد. در حالت کلی می توان گفت کاراکتر ۶ ام به بعد در حافظه مربوط به متغیر pass ذخیره می شود لذا اگر یک عبارت با حداقل ۶ کاراکتر به عنوان پسورد ورودی وارد شود که کاراکتر ۶ ام آن a نباشد با وجود اینکه این پسورد با مقدار "pass" برابر نبوده و برنامه وارد بلاک else بعد از دستور strcmp نمی شود اما به دلیل سرریز ، یک مقدار غیر صفر و نا برابر با a در داخل متغیر pass قرار می گیرد لذا با وجود آنکه عبارت wrong password چاپ می شود اما عبارت successful login چاپ می گردد. نکته قابل توجه دیگر این است که می دانیم در انتهای رشته همواره کاراکتر ۰ قرار خواهد گرفت و اگر یک پسورد ۵ حرفی داشته باشیم کاراکتر ۶ ام آن ۰ خواهد بود که خود این مقدار نیز سرریز خواهد کرد اما سرریز ۰ منجر به تغییر مقدار متغیر pass نمی گردد و همچنان مقدار ۰ در آن باقی خواهد ماند و خللی در اجرای برنامه اتفاق نمی افتد با وجود آنکه سرریز رخ داده است.

```
C:\Users\User\Desktop>a.exe

 Enter the password :
passaa

 Wrong Password

 Successful Login as Admin : )

C:\Users\User\Desktop>
```

```
C:\Users\User\Desktop>a.exe

 Enter the password :
password

 Wrong Password

 Successful Login

C:\Users\User\Desktop>
```

ب) تصویر زیر نتیجه اجرای برنامه کامپایل شده با فلگ -fno-stack-protector را نشان می دهد. دقیقا مشابه آنچه که در قسمت الف ذکر شد پس از ۵ کاراکتر سرریز رخ داده و به این سبب که کاراکتر ۶ ام یک کاراکتر ناصفر و مخالف با 'a' است داخل متغیر pass یک مقدار ناصفر و مخالف با 'a' دخیره می شود که منجر به چاپ عبارت successful login می گردد.

```
 Enter the password :
1234

 Wrong Password

 Failed to Login

C:\Users\User\Desktop>app.exe

 Enter the password :
12345

 Wrong Password

 Failed to Login

C:\Users\User\Desktop>app.exe

 Enter the password :
123456

 Wrong Password

 Successful Login

C:\Users\User\Desktop>app.exe
```

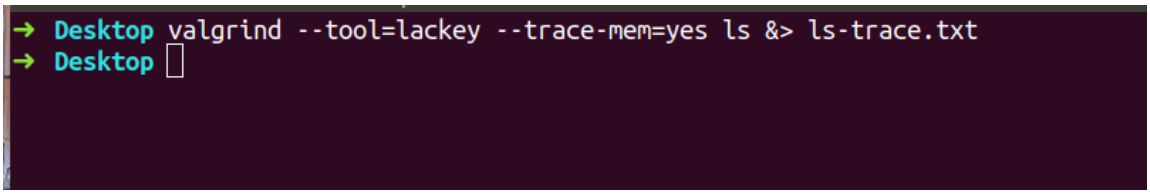
تصویر زیر نتیجه اجرای برنامه را پس از جایگزین شدن register int نشان می دهد. همانطور که مشخص است به نظر میرسد مقدار متغیر pass در نتیجه سرریز تغییر نکرده است همانطور که از نام رجیستر مشخص است به نظر میرسد این نوع تعریف متغیر منجر به تخصیص حافظه از پشته برای متغیر pass نمی شود بلکه یکی از رجیستر های پردازنده را به آن اختصاص می دهد و لذا سرریز buff در داخل خانه های بعدی پشته منجر به خلل در اجرای برنامه نمی گردد.

```
C:\Users\User\Desktop>gcc -g -fno-stack-protector x.c -o app

C:\Users\User\Desktop>app.exe

 Enter the password :
123456

 Wrong Password

 Failed to Login

C:\Users\User\Desktop>app.exe

 Enter the password :
1234567

 Wrong Password

 Failed to Login

C:\Users\User\Desktop>app.exe

 Enter the password :
12345678

 Wrong Password

 Failed to Login
```

ج) با استفاده از دستور fgets(buff, n, stdin) به جای دستور gets می توان حطر سرریز را برطرف کرد به علاوه لازم است تعداد کاراکتر های متغیر بافر را یک کاراکتر بیشتر در نظر بگیریم زیرا در حقیقت این دستور به تعداد n-1 کاراکتر از ورودی خوانده و در انتهای رشته ۰ را قرار می دهد سایر کاراکتر های ورودی را اسکیپ می کند لذا چون عبارت pass داری ۴ کاراکتر است اگر از buff[5] استفاده کنیم دستور fgets(buff, 5, stdin) خود به خود تنها ۴ کاراکتر از ورودی را برداشته و در انتهای آن ۰ قرار می دهد حال اگر کاربر عبارتی مانند 'password' را وارد کرده باشد خود به خود عبارت 'pass' داخل بافر ریخته شده و منجر به خطا خواهد شد لذا یک بیت بیشتر در نظر میگیرم تا درصورت ورود عبارت ناصحیح حتما عبارت ریخته شده در بافر با مقدار 'pass' متفاوت باشد. برنامه تصحیح شده تحت عنوان Q7-edited.c ذخیره شده است.

سوال ۸

الف) تصویر اجرا دستور ذکر شده در زیر آمده است.



بخشی از آدرس های بدست آمده در تصویر زیر قابل مشاهده است.

ب) آدرس های بدست آمده ۳۲ بیتی هستند به علاوه اگر سایز هر صفحه ۴ کیلو بایت باشد خواهیم داشت : $\frac{2^{32}}{2^{12}} = 2^{20}$ پس تعداد 2^{20} صفحه وجود دارد لذا ۲۰ بیت از آدرس مجازی متعلق به VPN می باشد پس برای بدست آوردن شماره صفحه مجازی هر آدرس کافی است ۲۰ بیت از سمت چپ جدا کرده و آن را به عدد دسیمال تبدیل کنیم. شماره صفحه مجازی مربوط به آدرس 0xf44ab8aa برابر $0xf44ab = 1000619$ و شماره صفحه مجازی مربوط به آدرس 0x044abd8a برابر $0x044ab = 17579$ می باشد.

ج) مطابق تفسیری که در قسمت قبل عنوان شد اسکریپت مورد نظر شماره صفحه مجازی نظیر هر آدرس را محاسبه می کند این تطابق را در تصویر زیر می توان مشاهده کرد.(اعداد سمت راست شماره صفحه مجازی نظیر ادرس های سمت چپ می باشند.)

```
I  04001090,3   16385
I  04001093,5   16385
 S 1ffefffc38,8 131055
I  04001ea0,1   16385
 S 1ffefffc30,8 131055
I  04001ea1,3   16385
I  04001ea4,2   16385
 S 1ffefffc28,8 131055
I  04001ea6,2   16385
 S 1ffefffc20,8 131055
I  04001ea8,2   16385
 S 1ffefffc18,8 131055
I  04001eaa,2   16385
 S 1ffefffc10,8 131055
I  04001eac,3   16385
I  04001eaf,1   16385
 S 1ffefffc08,8 131055
I  04001eb0,4   16385
I  04001eb4,2   16385
I  04001eb6,4   16385
I  04001eba,2   16385
I  04001ebc,3   16385
I  04001ebf,7   16385
 L 04227e68,8   16935
I  04001ec6,7   16385
 S 04227720,8   16935
```

د – ۱) مطابق تصویر زیر تعداد کل درخواست های حافظه برابر ۷۳۷۵۱۸ عدد می باشد.(از فایل ls-trace.txt خطوط اضافه ابتدا و انتهایی را حذف کرده سپس با برنامه wc تعداد خطوط (آدرس ها) ها شمرده ایم.)

```
→ sec_project wc ls-trace.txt
  737518  1475036 10498145 ls-trace.txt
→ sec_project ./va2vpn.py ls-trace.txt 737518
→ sec_project
```

د – ۲) ابتدا شماره صفحه مجازی نظیر هر آدرس را به کمک اسکریپت قسمت قبل بدست آورده سپس با دستور sort آن ها را مرتب سازی کرده و با استفاده از برنامه uniq تعداد صفحات مجزا را محاسبه می کنیم. طبق تصویر زیر تنها ۲۷۶ صفحه مجزا از حافظه درخواست شده اگر چه که تعداد آدرس های درخواستی ۷۳۷۵۱۸ مورد بوده است.

```
→ sec_project sort vpn.txt| uniq > uniq_vpn.txt
→ sec_project wc uniq_vpn.txt
 276  276 1606 uniq_vpn.txt
→ sec_project
```

CLOCK ⇒ ./paging-policy.py –addressfile=vpn.txt –policy=CLOCK –cachesize=3 -c
Hit Rate : 94.99

```
Access: 16417  MISS Left  -> [16408, 16935, 16417] <- Right Replaced:131056 [Hits:9482 Misses:499]
Access: 16408  HIT  Left  -> [16408, 16935, 16417] <- Right Replaced:- [Hits:9483 Misses:499]
Access: 131056  MISS Left  -> [16408, 16417, 131056] <- Right Replaced:16935 [Hits:9483 Misses:500]
Access: 16408  HIT  Left  -> [16408, 16417, 131056] <- Right Replaced:- [Hits:9484 Misses:500]
Access: 16408  HIT  Left  -> [16408, 16417, 131056] <- Right Replaced:- [Hits:9485 Misses:500]
Access: 16408  HIT  Left  -> [16408, 16417, 131056] <- Right Replaced:- [Hits:9486 Misses:500]
Access: 16408  HIT  Left  -> [16408, 16417, 131056] <- Right Replaced:- [Hits:9487 Misses:500]
Access: 16408  HIT  Left  -> [16408, 16417, 131056] <- Right Replaced:- [Hits:9488 Misses:500]
Access: 16408  HIT  Left  -> [16408, 16417, 131056] <- Right Replaced:- [Hits:9489 Misses:500]
Access: 16408  HIT  Left  -> [16408, 16417, 131056] <- Right Replaced:- [Hits:9490 Misses:500]
Access: 16408  HIT  Left  -> [16408, 16417, 131056] <- Right Replaced:- [Hits:9491 Misses:500]
Access: 16408  HIT  Left  -> [16408, 16417, 131056] <- Right Replaced:- [Hits:9492 Misses:500]
Access: 16408  HIT  Left  -> [16408, 16417, 131056] <- Right Replaced:- [Hits:9493 Misses:500]
Access: 16408  HIT  Left  -> [16408, 16417, 131056] <- Right Replaced:- [Hits:9494 Misses:500]
Access: 16935  MISS Left  -> [16408, 131056, 16935] <- Right Replaced:16417 [Hits:9494 Misses:501]
Access: 16408  HIT  Left  -> [16408, 131056, 16935] <- Right Replaced:- [Hits:9495 Misses:501]
Access: 16408  HIT  Left  -> [16408, 131056, 16935] <- Right Replaced:- [Hits:9496 Misses:501]
Access: 16935  HIT  Left  -> [16408, 131056, 16935] <- Right Replaced:- [Hits:9497 Misses:501]
Access: 16408  HIT  Left  -> [16408, 131056, 16935] <- Right Replaced:- [Hits:9498 Misses:501]
Access: 16408  HIT  Left  -> [16408, 131056, 16935] <- Right Replaced:- [Hits:9499 Misses:501]

FINALSTATS hits 9499   misses 501   hitrate 94.99
```

FIFO ⇒ ./paging-policy.py –addressfile=vpn.txt –policy=FIFO –cachesize=3 -c
Hit Rate : 92.72

```
Access: 16417  MISS FirstIn -> [16408, 16935, 16417] <- Lastin  Replaced:131056 [Hits:9256 Misses:725]
Access: 16408  HIT  FirstIn -> [16408, 16935, 16417] <- Lastin  Replaced:- [Hits:9257 Misses:725]
Access: 131056  MISS FirstIn -> [16935, 16417, 131056] <- Lastin  Replaced:16408 [Hits:9257 Misses:726]
Access: 16408  MISS FirstIn -> [16417, 131056, 16408] <- Lastin  Replaced:16935 [Hits:9257 Misses:727]
Access: 16408  HIT  FirstIn -> [16417, 131056, 16408] <- Lastin  Replaced:- [Hits:9258 Misses:727]
Access: 16408  HIT  FirstIn -> [16417, 131056, 16408] <- Lastin  Replaced:- [Hits:9259 Misses:727]
Access: 16408  HIT  FirstIn -> [16417, 131056, 16408] <- Lastin  Replaced:- [Hits:9260 Misses:727]
Access: 16408  HIT  FirstIn -> [16417, 131056, 16408] <- Lastin  Replaced:- [Hits:9261 Misses:727]
Access: 16408  HIT  FirstIn -> [16417, 131056, 16408] <- Lastin  Replaced:- [Hits:9262 Misses:727]
Access: 16408  HIT  FirstIn -> [16417, 131056, 16408] <- Lastin  Replaced:- [Hits:9263 Misses:727]
Access: 16408  HIT  FirstIn -> [16417, 131056, 16408] <- Lastin  Replaced:- [Hits:9264 Misses:727]
Access: 16408  HIT  FirstIn -> [16417, 131056, 16408] <- Lastin  Replaced:- [Hits:9265 Misses:727]
Access: 16408  HIT  FirstIn -> [16417, 131056, 16408] <- Lastin  Replaced:- [Hits:9266 Misses:727]
Access: 16408  HIT  FirstIn -> [16417, 131056, 16408] <- Lastin  Replaced:- [Hits:9267 Misses:727]
Access: 16935  MISS FirstIn -> [131056, 16408, 16935] <- Lastin  Replaced:16417 [Hits:9267 Misses:728]
Access: 16408  HIT  FirstIn -> [131056, 16408, 16935] <- Lastin  Replaced:- [Hits:9268 Misses:728]
Access: 16408  HIT  FirstIn -> [131056, 16408, 16935] <- Lastin  Replaced:- [Hits:9269 Misses:728]
Access: 16935  HIT  FirstIn -> [131056, 16408, 16935] <- Lastin  Replaced:- [Hits:9270 Misses:728]
Access: 16408  HIT  FirstIn -> [131056, 16408, 16935] <- Lastin  Replaced:- [Hits:9271 Misses:728]
Access: 16408  HIT  FirstIn -> [131056, 16408, 16935] <- Lastin  Replaced:- [Hits:9272 Misses:728]

FINALSTATS hits 9272   misses 728   hitrate 92.72
```

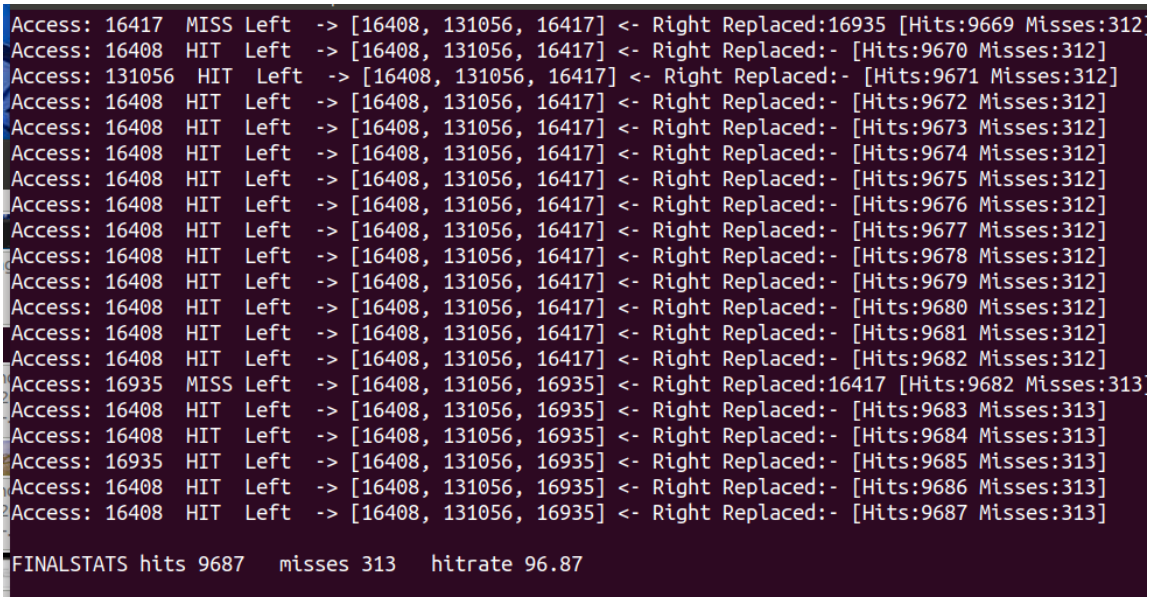
LRU ⇒ ./paging-policy.py –addressfile=vpn.txt –policy=LRU –cachesize=3 -c
Hit Rate : 94.35

```
Access: 16417  MISS LRU -> [16935, 16408, 16417] <- MRU Replaced:131056 [Hits:9418 Misses:563]
Access: 16408  HIT  LRU -> [16935, 16417, 16408] <- MRU Replaced:- [Hits:9419 Misses:563]
Access: 131056  MISS LRU -> [16417, 16408, 131056] <- MRU Replaced:16935 [Hits:9419 Misses:564]
Access: 16408  HIT  LRU -> [16417, 131056, 16408] <- MRU Replaced:- [Hits:9420 Misses:564]
Access: 16408  HIT  LRU -> [16417, 131056, 16408] <- MRU Replaced:- [Hits:9421 Misses:564]
Access: 16408  HIT  LRU -> [16417, 131056, 16408] <- MRU Replaced:- [Hits:9422 Misses:564]
Access: 16408  HIT  LRU -> [16417, 131056, 16408] <- MRU Replaced:- [Hits:9423 Misses:564]
Access: 16408  HIT  LRU -> [16417, 131056, 16408] <- MRU Replaced:- [Hits:9424 Misses:564]
Access: 16408  HIT  LRU -> [16417, 131056, 16408] <- MRU Replaced:- [Hits:9425 Misses:564]
Access: 16408  HIT  LRU -> [16417, 131056, 16408] <- MRU Replaced:- [Hits:9426 Misses:564]
Access: 16408  HIT  LRU -> [16417, 131056, 16408] <- MRU Replaced:- [Hits:9427 Misses:564]
Access: 16408  HIT  LRU -> [16417, 131056, 16408] <- MRU Replaced:- [Hits:9428 Misses:564]
Access: 16408  HIT  LRU -> [16417, 131056, 16408] <- MRU Replaced:- [Hits:9429 Misses:564]
Access: 16408  HIT  LRU -> [16417, 131056, 16408] <- MRU Replaced:- [Hits:9430 Misses:564]
Access: 16935  MISS LRU -> [131056, 16408, 16935] <- MRU Replaced:16417 [Hits:9430 Misses:565]
Access: 16408  HIT  LRU -> [131056, 16935, 16408] <- MRU Replaced:- [Hits:9431 Misses:565]
Access: 16408  HIT  LRU -> [131056, 16935, 16408] <- MRU Replaced:- [Hits:9432 Misses:565]
Access: 16935  HIT  LRU -> [131056, 16408, 16935] <- MRU Replaced:- [Hits:9433 Misses:565]
Access: 16408  HIT  LRU -> [131056, 16935, 16408] <- MRU Replaced:- [Hits:9434 Misses:565]
Access: 16408  HIT  LRU -> [131056, 16935, 16408] <- MRU Replaced:- [Hits:9435 Misses:565]

FINALSTATS hits 9435   misses 565   hitrate 94.35
```

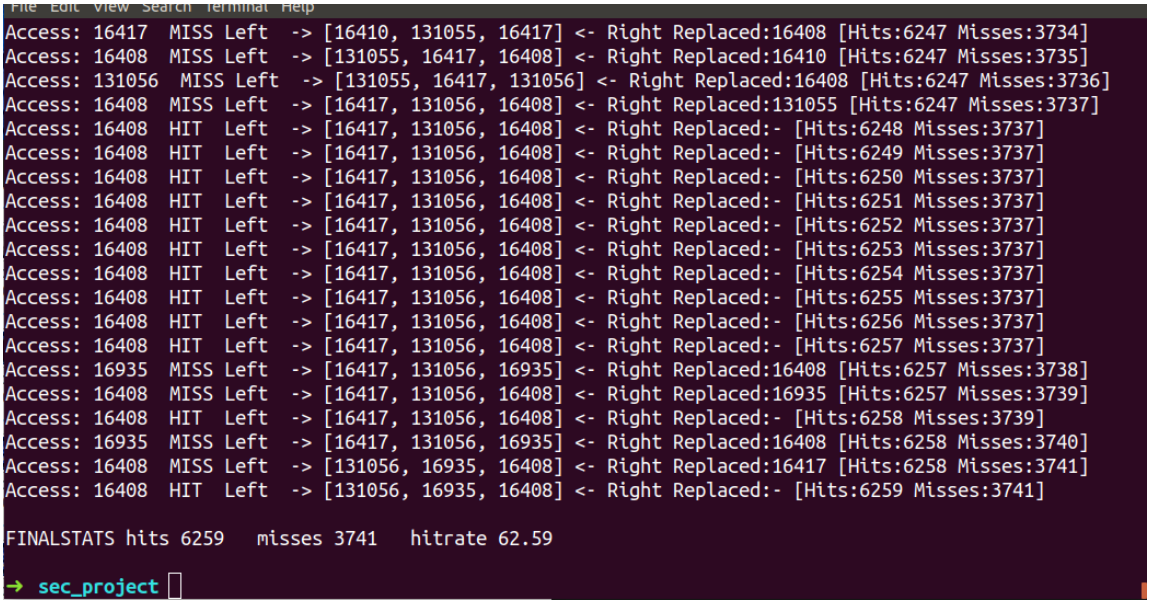
RAND ⇒ ./paging-policy.py –addressfile=vpn.txt –policy=RAND –cachesize=3 -c
Hit Rate : 94.27

```
Access: 16417  MISS Left  -> [16935, 131056, 16417] <- Right Replaced:16408 [Hits:9409 Misses:572]
Access: 16408  MISS Left  -> [16935, 131056, 16408] <- Right Replaced:16417 [Hits:9409 Misses:573]
Access: 131056  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9410 Misses:573]
Access: 16408  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9411 Misses:573]
Access: 16408  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9412 Misses:573]
Access: 16408  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9413 Misses:573]
Access: 16408  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9414 Misses:573]
Access: 16408  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9415 Misses:573]
Access: 16408  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9416 Misses:573]
Access: 16408  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9417 Misses:573]
Access: 16408  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9418 Misses:573]
Access: 16408  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9419 Misses:573]
Access: 16408  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9420 Misses:573]
Access: 16408  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9421 Misses:573]
Access: 16935  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9422 Misses:573]
Access: 16408  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9423 Misses:573]
Access: 16408  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9424 Misses:573]
Access: 16935  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9425 Misses:573]
Access: 16408  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9426 Misses:573]
Access: 16408  HIT  Left  -> [16935, 131056, 16408] <- Right Replaced:- [Hits:9427 Misses:573]

FINALSTATS hits 9427   misses 573   hitrate 94.27
```

OPT ⇒ ./paging-policy.py –addressfile=vpn.txt –policy=OPT –cachesize=3 -c
Hit Rate : 96.87



UNOPT ⇒ ./paging-policy.py –addressfile=vpn.txt –policy=UNOPT –cachesize=3 -c
Hit Rate : 62.59



د – ۴) تعداد ۱۰۰۰۰ آدرس اول فایل VPN.txt به عنوان ورودی به شبیه ساز داده شده است مطابق نمودار زیر می توان چندین نتیجه گیری کرد:

هرچه سایز cache بزرگتر باشد Hit Rate نیز بیشتر خواهد بود.
دو سیاست جایگزینی LRU , CLOCK تا حد زیادی شبیه به حالت بهینه عمل می کنند با وجود آنکه پیاده سازی آن ها راحت تر است و نیازی به دانستن آینده نیز نمی باشد به مراتب پیاده سازی CLOCK از LRU نیز ساده تر می باشد.
در مقایسه با ۵ سیاست به جز UNOPT سیاست FIFO کارآیی کمتری داشته است.
در بدترین شرایط میزان Hit Rate حدود ۶۲ درصد بوده است که با وجود اینکه کمترین مقدار است همچنان قابل قبول بوده و مزیت استفاده از cache را نمایان می سازد.