Lista 6

tags: SO

Zadanie 1

Zadanie 1. W każdym z poniższych przypadków zakładamy, że początkowa **tożsa**
to: `ruid=1000, euid=0, suid=0`. Jak zmieni się tożsamość procesu po wywołaniu
(a) `setuid(2000)`, (b) `setreuid(-1, 2000)`, (c) `seteuid(2000)`, (d) `setresu`
Odpowiedź uzasadnij posługując się podręcznikami systemowymi `setuid(2)`, `setreu`

Czy proces z tożsamością `ruid=0, euid=1000, suid=1000` jest uprzywilejowany? O

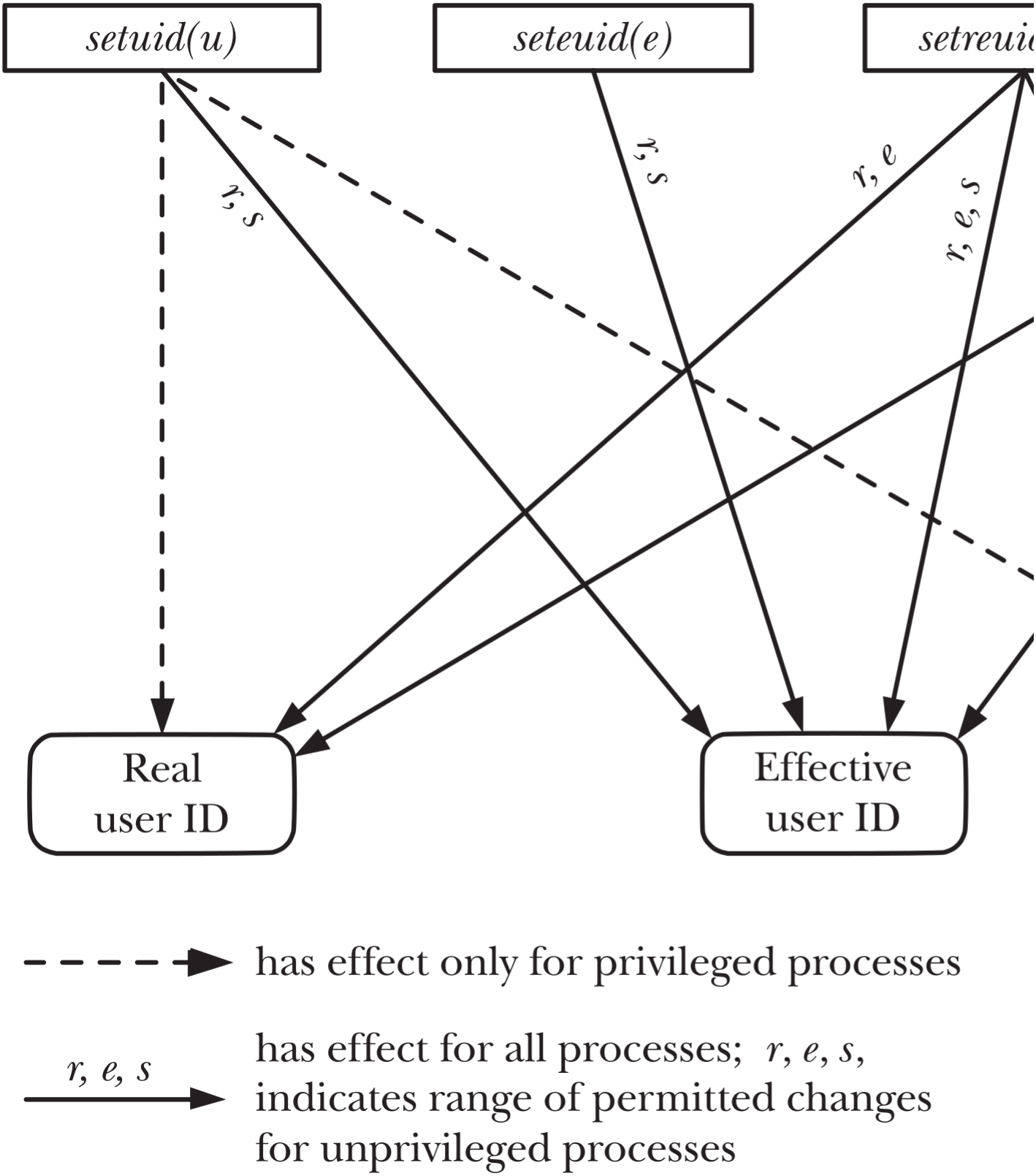
::: info **tożsamość** – łącznie rzeczywisty(ruid), efektywny(euid), saved(suid) UID i file-system ID (oraz ich odpowiedniki dla grup) **proces uprzywilejowany** – proces o euid równym 0 :::

Every process has a set of associated numeric user identifiers (UIDs) and group identifiers (GIDs). Sometimes, these are referred to as process credentials. These identifiers are as follows:

- real user ID and group ID;
- effective user ID and group ID;
- saved set-user-ID and saved set-group-ID;
- file-system user ID and group ID (Linux-specific); and
- supplementary group IDs.

In this chapter, we look in detail at the purpose of these process identifiers and describe the system calls and library functions that can be used to retrieve and change them. We also discuss the notion of privileged and unprivileged processes, and the use of the set-user-ID and set-group-ID mechanisms, which allow the creation of programs that run with the privileges of a specified user or group.

Interface	Purpose and effect within:	
	unprivileged process	**privileg**
setuid(u) *setgid(g)*	Change effective ID to the same value as current real or saved set ID	Change effective saved set any (sing
seteuid(e) *setegid(e)*	Change effective ID to the same value as current real or saved set ID	Change ID to any
setreuid(r, e) *setregid(r, e)*	(Independently) change real ID to same value as current real or effective ID, and effective ID to same value as current real, effective, or saved set ID	(Indepen change r effective any value
setresuid(r, e, s) *setresgid(r, e, s)*	(Independently) change real, effective, and saved set IDs to same value as current real, effective, or saved set ID	(Indepen change r effective, saved set any value



Jak zmieni się tożsamość po wywołaniu następujących funkcji?

a) setuid(2000) -> `ruid = 2000, euid = 2000, suid = 2000`

b) setreuid(-1, 2000) -> `ruid = 1000, euid = 2000, suid = 2000`

:::spoiler Jeśli `r` = -1 to `ruid` pozostaje takie samo. Zmieniając `euid` ustawiamy również `suid` na tą samą wartość :::

c) seteuid(2000) -> `ruid = 1000, euid = 2000, suid = 0`

:::spoiler Jako proces uprzywilejowany ustawiamy `euid` na 2000. Proces nieuprzywilejowany może jedynie przełączać się między wartościami `ruid` i `suid`. :::

d) setresuid(-1, 2000, 3000) -> `ruid = 1000, euid = 2000, suid = 3000`

Czy proces ruid=0 euid=1000 suid=1000 jest uprzywilejowany?

Nie jest, ponieważ `euid` != 0.

Zadanie 2

Zadanie 2. Jaką rolę pełnią bity uprawnień «rwx» dla katalogów w systemach u
czenie bitów «set-gid» i «sticky» dla katalogów. Napisz w pseudokodzie i zre
my_access(struct stat *sb, int mode)». Pierwszy i drugi argument opisano o
i access(2). Dla procesu o tożsamości zadanej przez getuid(2) i getgroups(2) p
sprawdza czy proces ma **upoważniony** dostęp «mode» do pliku o metadanych wczyt

Wskazówka: Rozważ uprawnienia katalogu «/usr/local» i «/tmp».

::: warning **r** - zezwala na odczyt zawartości katalogu (nazwy plików bez ich zawartości i metadanych) **w** - zezwala na modyfikację zawartości katalogu (przykładowo możemy wtedy zmieniać nazwy plików) **x** - zezwala na odczyt metadanych plików **set-gid** - sprawia, że utworzone pliki dziedziczą `guid` katalogu (w przeciwieństwie do dzieidziczenia po użytkowniku) **sticky** - plik może usunąć lub przemianować tylko właściciel katalogu lub pliku :::

```
struct stat {
   dev_t     st_dev;         /* ID of device containing file */
   ino_t     st_ino;         /* Inode number */
   mode_t    st_mode;        /* File type and mode */
   nlink_t   st_nlink;       /* Number of hard links */
   uid_t     st_uid;         /* User ID of owner */
   gid_t     st_gid;         /* Group ID of owner */
   dev_t     st_rdev;        /* Device ID (if special file) */
   off_t     st_size;        /* Total size, in bytes */
   blksize_t st_blksize;     /* Block size for filesystem I/O */
   blkcnt_t  st_blocks;      /* Number of 512B blocks allocated */
   struct timespec st_atim;  /* Time of last access */
   struct timespec st_mtim;  /* Time of last modification */
   struct timespec st_ctim;  /* Time of last status change */
};
```

```
 The mode specifies the accessibility check(s) to be performed, and is either the value F_OK,
or  a  mask consisting of the bitwise OR of one or more of R_OK, W_OK, and X_OK.  F_OK tests
for the existence of the file.  R_OK, W_OK, and X_OK test whether the file exists and grants
read, write, and execute permissions, respectively.

The  check  is  done using the calling process's real UID and GID, rather than the effective
IDs as is done when actually attempting an operation (e.g., open(2))  on  the  file.   Simi-
larly,  for  the root user, the check uses the set of permitted capabilities rather than the
set of effective capabilities; and for non-root users, the check uses an empty set of  capa-
bilities.

This allows set-user-ID programs and capability-endowed programs to easily determine the in-
voking user's authority.  In other words, access() does not answer the "can I read/write/ex-
ecute  this file?" question.  It answers a slightly different question: "(assuming I'm a se-
tuid binary) can the user who invoked me read/write/execute this file?",  which  gives  set-
user-ID  programs the possibility to prevent malicious users from causing them to read files
which users shouldn't be able to read.

If the calling process is privileged (i.e., its real UID is zero), then  an  X_OK  check  is
successful  for  a  regular file if execute permission is enabled for any of the file owner,
group, or other.
```

```
 getgroups()  returns the supplementary group IDs of
the calling process in  list.   The  argument  size
should  be  set to the maximum number of items that
can be stored in the buffer pointed to by list.  If
the  calling  process is a member of more than size
supplementary groups, then an error results.

If  size is zero, list is not modified, but the to-
tal number  of  supplementary  group  IDs  for  the
process is returned.  This allows the caller to de-
termine the size of a dynamically allocated list to
be used in a further call to getgroups().
```

bity dostępu: https://www.gnu.org/software/libc/manual/html_node/Permission-Bits.html

```
bool my_access(struct stat *sb, int mode)
    uid = getuid()

    if uid == 0
        return true

    if uid ma odpowiednie zdolności
        return true

    if uid == sb->st_uid
        return mode in sb->st_mode

    groups = getgroups()
    for group in groups
        if sb->st_gid == group
            return mode in mode(group)

    return mode in mode(others)
```

:::spoiler

```
bool my_access(struct stat *sb, int mode)
    uid = getuid()
    if (uid == 0)
        return true

    if (uid == sd->st_uid)
        return mode in sd->st_mode

    size = getgroups(0, NULL) // liczba grup
    groups[size]
    getgroups(groups, size)
    for group in groups
        if (sb->st_gid == group)
            return mode in group->mode

    return false
```

:::

Zadanie 3

Zadanie 3. Właścicielem pliku programu `su(1)` jest «root», a plik ma ustawion tożsamość będzie miał na początku proces wykonujący «su», jeśli przed `execve(2)`

Zreferuj działanie uproszczonej wersji programu su[1] zakładając, że wszystkie wywoł się bez błędów, a użytkownik zdołał się **uwierzytelnić**. Skoncentruj się na funkcjach c użytkowników, odczytujących i sprawdzających hasło, oraz zmieniających tożsamość

Zadanie 4

Zadanie 4. Na podstawie §38.2 i §38.3 wyjaśnij czemu programy uprzywilejowane na sposób, by operowały z najmniejszym możliwym zestawem upoważnień (ang. *the le* wytyczne dotyczące projektowania takich programów. Zapoznaj się z §39.1 i wytłum zestaw funkcji systemu uniksowego do implementacji programów uprzywilejowanyc Jak starają się to naprawić zdolności (ang. *capabilities*)? Dla nieuprzywilejowanego zdolności «CAP_DAC_READ_SEARCH» i «CAP_KILL» jądro pomija sprawdzanie upow pewnych akcji – wymień je. Kiedy proces użytkownika może wysłać sygnał do inneg

Na podstawie §38.2 i §38.3 wyjaśnij czemu programy uprzywilejowane należy projektować w taki sposób, by operowały z najmniejszym możliwym zestawem upoważnień (ang.the least privilege).

- Minimalne uprzywilejowanie ogranicza szkody jakie mógłby wyrządzić błędnie działający program (np. program modyfikujący pliki w systemie mający uprawnienia roota)
- Wiele programów jest podatnych na ataki (między innymi przez nadpisanie stosu), więc posiadanie przez program niepotrzebnych uprawnień może zwiększać ryzyko szkodliwego ataku.
- Przed wywołaniem exec należy przygotować poziom uprawnień procesu tak, aby nowy program powstał z minimalnym zestawem upoważnień oraz nie mógł odzyskać poprzenich (tzn. potrzebny jest reset). Przykładowo proces mógłby wykonać exec do powłoki i (przy odpowiednio wysokich uprawnieniach) wyrządzić szkodę w systemie.
- Trzeba też pamiętać o usuwaniu uprawnień przez zamykanie deskryptorów plików przy uruchamianiu innych procesów. Uprzywilejowany program może otworzyć plik, do którego normalne procesy nie mają dostępu. Wynikowy deskryptor otwartego pliku reprezentuje uprzywilejowany zasób.

Wytyczne dotyczące projektowania takich programów

- Program powinien zacząć pracę z możliwie najmniejszymi uprawnieniami. Dobrą praktyką jest wyzbycie się uprawnień programu i tymczasowe przywrócanie ich, gdy są potrzebne.

```
uid_t orig_euid;

orig_euid = geteuid();
if (seteuid(getuid()) == -1)            /* Drop privileges */
    errExit("seteuid");

/* Do unprivileged work */

if (seteuid(orig_euid) == -1)           /* Reacquire privileges */
    errExit("seteuid");

/* Do privileged work */
```

- Jeśli dany poziom uprawnień nie będzie już nigdy potrzebny należy je permanentnie usunąć wykonując tzw. reset uprawnień, czyli przypisanie wszystkim ID grup i użytkowników odpowiadającym im wartością rzeczywistym.
- przed wywołaniem execv należy ustawić odpowiednie uprawnienia, upewnić się, że wszystkie identyfikatory użytkowników (grup) procesów są resetowane do tej samej wartości, co rzeczywisty identyfikator użytkownika (grupy), tak aby nowy program nie startował z uprawnieniami jakich mieć nie powinien, a także żeby nie mógł ich odzyskać
- Należy również zadbać o stosowny poziom uprawnień przy wykonywaniu exec (tak jak opisano wcześniej)

Czemu standardowy zestaw funkcji systemu uniksowego do implementacji programów uprzywilejowanych jest niewystarczający.

W sytuacji, w której chcemy, aby dany proces wykonał zadanie, do którego uprawnienia ma tylko *root* to musimy uczynić ten proces procesem uprzywilejowanym. Niestety takie rozwiązanie nadaje procesowi uprawnienia również do wykonywania operacji niezwiązanych z interesującym nas zadaniem, zatem takie podejście nie przestrzega wytycznej nadawania minimalnych w kontekście wykonywanego zadania uprawnień.

Jak starają się to naprawić zdolności (ang.capabilities)? Uprawnienia *roota* są dzielone na zdolności. Możemy wtedy przekazać procesowi tylko te zdolności, które są niezbędne do wykonania zadania.

Dla nieuprzywilejowanego procesu posiadającego zdolności «CAP_DAC_READ_SEARCH» i «CAP_KILL» jądro pomija sprawdzanie upoważnień do wykonywaniapewnych akcji – wymień je.

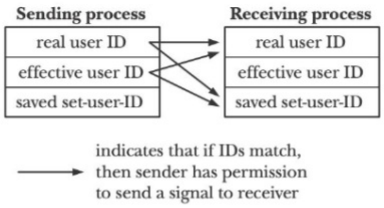
- **CAP_DAC_READ_SEARCH** - Pomijanie sprawdzania uprawnień do odczytu plików oraz odczytu katalogu i uruchamiania programów.
- **CAP_KILL** - Pomijanie sprawdzania uprawnień do wysyłania sygnałów.

Kiedy proces użytkownika może wysłać sygnał do innego procesu?

Proces ma uparawnienia do wsyłania sygnału, gdy:

- jest uprzywilejowany (CAP_KILL)
- jego `ruid` lub `euid` jest równy `ruid` lub `suid` procesu docelowego

Upoważnienie: wysyłanie sygnałów (LPI 20-5)



Dotyczy **wyłącznie** sygnałów wysyłanych przy pomocy `kill(2)`. Jądro jest uprzywilejowane, tak jak użytkownik `root`, więc zawsze może wysyłać sygnał do procesu (np. `SIGTSTP` albo `SIGSEGV`).

Jeśli sprawdzenie według powyższej tabelki nie wyjdzie → `EPERM`. Wyjątkiem jest `SIGCONT` w obrębie tej samej sesji. **Dlaczego?**

- wysyłany sygnał to *SIGCONT* i oba procesy są w tej samej sesji.

Zadanie 5

Zadanie 5. Jakie zadania pełni procedura `exit(3)` z biblioteki standardowej? Opisz p plików, które mogą wystąpić dla strumieni biblioteki `stdio(3)` w przypadku uży `execve(2)` i `_exit(2)`. Jak zapobiec tym problemom? Jaka jest domyślna strategia związanego z (a) plikiem terminala (b) plikiem zwykłym (c) standardowym wyjścier

Piszesz program który używa biblioteki «`stdio`». Działanie programu da się przerw Ma on wtedy opróżnić wszystkie bufory otwartych strumieni i dopiero wtedy wyjść. pamiętając, że w procedurach obsługi sygnału nie wolno korzystać z funkcji, które r

Co robi exit?

```
 The  exit() function causes normal process termination and the value of
status & 0xFF is returned to the parent (see wait(2)).

All functions registered with atexit(3) and on_exit(3) are  called,  in
the  reverse  order  of their registration.

All open stdio(3) streams are flushed and  closed.   Files  created  by
tmpfile(3) are removed.
```

Problemy które mogą wystąpić podczas:

Fork

Jeśli bufor jest niepusty to jego zawartość może zostać użyta wielokrotnie, np.:

```
 printf("foo");
fork();
```

_exit, execve

Wywołując te procedury utracimy zgromadzone w buforze dane.

Jak zapobiec tym problemom?

Możemy wyłączyć buforowanie za pomocą `setbuf`, wykonać `fflush` opróżniając bufor lub dodać znak końca linii (jeśli dane buforowane są wierszami).

Strategie buforowania: a) pliki terminala - buforowanie wierszami b) pliki dyskowe - buforowanie pełne (dopóki bufor nie jest zapełniony) c) *stderr* - niebuforowane

---

abort	faccessat	linkat	select	socketpair
accept	fchmod	listen	sem_post	stat
access	fchmodat	lseek	send	symlink
aio_error	fchown	lstat	sendmsg	symlinkat
aio_return	fchownat	mkdir	sendto	tcdrain
aio_suspend	fcntl	mkdirat	setgid	tcflow
alarm	fdatasync	mkfifo	setpgid	tcflush
bind	fexecve	mkfifoat	setsid	tcgetattr
cfgetispeed	fork	mknod	setsockopt	tcgetpgrp
cfgetospeed	fstat	mknodat	setuid	tcsendbreak
cfsetispeed	fstatat	open	shutdown	tcsetattr
cfsetospeed	fsync	openat	sigaction	tcsetpgrp
chdir	ftruncate	pause	sigaddset	time
chmod	futimens	pipe	sigdelset	timer_getoverrun
chown	getegid	poll	sigemptyset	timer_gettime
clock_gettime	geteuid	posix_trace_event	sigfillset	timer_settime
close	getgid	pselect	sigismember	times
connect	getgroups	raise	signal	umask
creat	getpeername	read	sigpause	uname
dup	getpgrp	readlink	sigpending	unlink
dup2	getpid	readlinkat	sigprocmask	unlinkat
execl	getppid	recv	sigqueue	utime
execle	getsockname	recvfrom	sigset	utimensat
execv	getsockopt	recvmsg	sigsuspend	utimes
execve	getuid	rename	sleep	wait
_Exit	kill	renameat	sockatmark	waitpid
_exit	link	rmdir	socket	write

Funkcje bezpieczne w procedurach obsługi sygnału:

Jak widać należy do nich `tcflush`, którego możemy użyć do opróżnienia buforów, a następnie wyjść procedurą `_exit`.

Zadanie 6

Ściągnij ze strony przedmiotu archiwum «so21_lista_6.tar.gz», następnie rozpakuj i zapoznaj si

UWAGA! Można modyfikować tylko te fragmenty programów, które zostały oznaczone w koment

Zadanie 6 (2). Program «writeperf» służy do testowania wydajności operacji microbenchmark[2] wczytuje z linii poleceń opcje i argumenty opisane dalej. Na stanc t trójkątów (opcja «-t») prostokątnych o boku złożonym z l znaków gwiazdki standardowe wyjście zostało przekierowane do pliku oraz została podana opcja «-s» programu bufory pliku zostaną zsynchronizowane z dyskiem wywołaniem fsync(2)

Program realizuje pięć wariantów zapisu do pliku:

- Każdą linię trójkąta zapisuje osobno wywołaniem write(2) (argument «wri
- Używa strumienia biblioteki stdio bez buforowania (argument «fwrite»), z (argument «fwrite-line») i **buforowaniem pełnym** (argument «fwrite-f
- Wykorzystuje wywołanie systemowe writev(2) do zapisania do «IOV_MAX» l

Twoim zadaniem jest odpowiednie skonfigurowanie bufora strumienia «stdout» setvbuf(3) oraz zaimplementowanie metody zapisu z użyciem «writev».

Przy pomocy skryptu powłoki «writeperf.sh» porównaj wydajność wymienionych Uzasadnij przedstawione wyniki. Miej na uwadze liczbę wywołań systemowych (nale się narzędziem strace(1) z opcją «-c») oraz liczbę kopii danych wykonanych cele linii do buforów dysku.

```
ssize_t writev(int fd, const struct iovec *iov, int iovcnt);

The writev() system call writes iovcnt buffers of data described by iov to the
ated with the file descriptor fd ("gather output").

The pointer iov points to an array of iovec structures, defined in <sys/uio.h>

    struct iovec {
        void  *iov_base;    /* Starting address */
        size_t iov_len;     /* Number of bytes to transfer */
    };
```

```
/* TODO: Attach new buffer to stdout stream. */
    buf = malloc(size * sizeof(char));
    setvbuf(stdout, buf, mode, size);
```

```
/* TODO: Write file by filling in iov array and issuing writev. */

    int i = 0;
    for (int j = 0; j < times; j++)
      for (int k = 0; k < length; k++) {

        if (i == n) {
          Writev(STDOUT_FILENO, iov, n);
          i = 0;
        }

        iov[i].iov_base = line + k;
        iov[i].iov_len = length + 1 - k;
        i++;
      }
    Writev(STDOUT_FILENO, iov, i);
```

```
Method: write

real    0m2.044s
user    0m0.032s
sys     0m2.000s
594c417685170dd3eb60286c0f634dc9  test

Method: fwrite

real    0m2.029s
user    0m0.109s
sys     0m1.917s
594c417685170dd3eb60286c0f634dc9  test

Method: fwrite-line

real    0m2.098s
user    0m0.128s
sys     0m1.966s
594c417685170dd3eb60286c0f634dc9  test

Method: fwrite-full

real    0m0.310s
user    0m0.036s
sys     0m0.273s
594c417685170dd3eb60286c0f634dc9  test

Method: writev

real    0m0.169s
user    0m0.004s
sys     0m0.164s
594c417685170dd3eb60286c0f634dc9  test
```

Zadanie 7

Zadanie 7. Program «id» drukuje na standardowe wyjście **tożsamość**, z którą zos

```
1 $ id
2 uid=1000(cahir) gid=1000(cahir) groups=1000(cahir),20(dialout),24(cdrom),25(f
3 27(sudo),29(audio),30(dip),44(video),46(plugdev),108(netdev),123(vboxusers),9
```

Uzupełnij procedurę «getid» tak by zwracała identyfikator użytkownika getuid(getgid(2) oraz tablicę identyfikatorów i liczbę grup dodatkowych getgroups(2). Ni liczby grup, do których należy użytkownik. Dlatego należy stopniowo zwiększać rozm pomocy realloc(3), aż pomieści rezultat wywołania «getgroups». Należy również u «uidname» i «gidname» korzystając odpowiednio z getpwuid(3) i getgrgid(3).

The **getpwuid**() function returns a pointer to a structure containing the broken-
the record in the password database that matches the user ID uid.

The passwd structure is defined in <pwd.h> as follows:

```
    struct passwd {
        char   *pw_name;       /* username */
        char   *pw_passwd;     /* user password */
        uid_t   pw_uid;        /* user ID */
        gid_t   pw_gid;        /* group ID */
        char   *pw_gecos;      /* user information */
        char   *pw_dir;        /* home directory */
        char   *pw_shell;      /* shell program */
    };
```

The **getgrgid**() function returns a pointer to a structure containing the broken
the record in the group database that matches the group ID gid.

The group structure is defined in <grp.h> as follows:

```
    struct group {
        char   *gr_name;        /* group name */
        char   *gr_passwd;      /* group password */
        gid_t   gr_gid;         /* group ID */
        char  **gr_mem;         /* NULL-terminated array of pointers
                                   to names of group members */
    };
```

```
static const char *uidname(uid_t uid) {
  /* TODO: Something is missing here! */
  return getpwuid(uid)->pw_name;
}

static const char *gidname(gid_t gid) {
  /* TODO: Something is missing here! */
  return getgrgid(gid)->gr_name;
}
```

```
static int getid(uid_t *uid_p, gid_t *gid_p, gid_t **gids_p) {
  gid_t *gids = NULL;
  int ngid = 2;
  int groups;

  /* TODO: Something is missing here! */

  *gids_p = gids;
  return groups;
}
```