Systemy operacyjne

Laboratorium 3

Mateusz Małek
24 marca 2017

Laboratorium 3

Tworzenie procesów, środowisko procesu, sterowanie procesami

Zmienne środowiskowe

- Przechowywane jako lista stringów postaci "NAZWA=WARTOSC"
- Jak dostać się do wszystkich zmiennych ze środowiska programu?
  - Sposób standardowy i przenośny (POSIX):
    - W kodzie umieszczamy: `extern char** environ; ...`
    - … a następnie odwołujemy się do tablicy `environ`
  - Sposób nieustandaryzowany, ale powszechnie dostępny:
    - Trójargumentowy main: `int main(int argc, char*[] argv, char*[] envp)`
    - Korzystanie identyczne jak z tablicy `environ`; de facto jako trzeci argument zostaje przekazana właśnie tablica `environ`
  - W obu przypadkach dostajemy zmienne środowiskowe w postaci wspomnianej listy stringów
- Jeśli chcemy otrzymać wartość konkretnej zmiennej, to są ku temu lepsze sposoby

Zmienne środowiskowe

- char *getenv(const char *name) - man 3 getenv
  - Zwraca wskaźnik do (tekstowej!) wartości zmiennej środowiskowej name lub NULLa w sytuacji kiedy ta zmienna nie jest ustawiona
  - Wartości znajdującej się pod zwróconym wskaźnikiem nie należy modyfikować, a uzyskanego wskaźnika nie należy zapamiętywać!
    - Wariant optymistyczny: zmieniając stringa, zmienimy wartość zmiennej środowiskowej
    - Wariant pesymistyczny: przy kolejnym wywołaniu getenv nasza “zapisana” zmienna zmieni swoją wartość...
  - W programach typu SUID/SGID (lub korzystających z mechanizmu “capabilities”) do pobierania wartości “wrażliwych” zmiennych środowiskowych należy używać funkcji char *secure_getenv(const char *name)

Zmienne środowiskowe

- int setenv(const char *name, const char *value, int overwrite)
  - Dla overwrite = 0: o ile zmienna name nie jest już ustawiona, to dodaje do środowiska zmienną name o wartości value (w przeciwnym razie nic nie robi)
  - Dla overwrite != 0: bezwarunkowo ustawia wartość zmiennej środowiskowej name na wartość value
  - name nie może być NULLem i nie może zawierać znaku =
  - Usunięcie zmiennej środowiskowej to coś innego niż ustawienie jej wartości na pusty string!
- int unsetenv(const char *name)
  - O ile zmienna name jest ustawiona, to usuwa ją ze środowiska programu (w przeciwnym razie nic nie robi)
- Obie funkcje opisane w man 3 setenv

Zmienne środowiskowe

- int putenv(char *string) - man 3 putenv
  - Pozwala ustawić, zmienić lub usunąć wartość zmiennej środowiskowej
  - Jeśli otrzyma argument postaci "NAZWA=WARTOSC", to dodaje go do środowiska programu (ewentualnie zastępując istniejącą zmienną)
    - Dokładnie tak - to nie jest ustawienie zmiennej NAZWA na wartość WARTOSC!
      - **Za dokumentacją:** “The pointer string given to putenv() is used”
      - **Konsekwencja:** nie wolno przekazywać jako argumentu zmiennych lokalnych lub dynamicznie zaalokowanych stringów które później zwalniamy
      - **Konsekwencja 2:** po użyciu putenv zmiany stringa użytego jako argument powodują zmianę wartości zmiennej środowiskowej!
      - … chyba że korzystamy z glibc w wersjach od 2.0 do 2.1.1, gdzie string przekazany do putenv kopiowano przed ustawieniem (wyciek pamięci gratis)
  - Jeśli otrzyma argument postaci "NAZWA", to usuwa ze środowiska programu zmienną NAZWA (lub nic nie robi, jeśli nie istniała)

Zmienne środowiskowe

- int clearenv(void) - man 3 clearenv
  - Usuwa wszystkie zmienne ze środowiska programu
  - Przydaje się w sytuacjach, w których nowy program lub proces potomny chcemy uruchomić w środowisku zawierającym tylko konkretnie, jawnie ustawione zmienne

Nowe procesy

- int clone(dużo argumentów…) - man 2 clone
  - Pozwala na tworzenie procesów potomnych, które trochę zasobów i środowiska dzielą z rodzicem, a trochę nie…
  - “Brama” do mechanizmu namespaces, używanego m.in. przez LXC i Dockera
- pid_t fork(void) - man 2 fork
  - “Rozwidla” program na proces macierzysty i proces potomny
  - “The child process is an exact duplicate of the parent process” - w miejscu wywołania fork powstaje drugi (prawie) identyczny proces, który wykonuje się od linijki w której nastąpił fork
    - “... except for the following points” - w dokumentacji opis czego potomek nie dziedziczy
  - W procesie macierzystym funkcja zwróci numer PID procesu potomnego
  - W procesie potomnym funkcja zwróci wartość 0
  - Trzeba wynik fork przypisać do zmiennej i zrobić if-a (> 0 - rodzic, == 0 - potomek, < 0 - błąd)
  - Rodzic (zwykle) powinien poczekać na zakończenie procesów potomnych funkcjami z rodziny wait

Oczekiwanie na procesy potomne

- pid_t wait(int *wstatus)
  - Równoważne wywołaniu waitpid(-1, &wstatus, 0)
  - “Zawisa” w oczekiwaniu na zakończenie się któregoś z jeszcze działających procesów potomnych, po czym zwraca PID zakończonego procesu, a do miejsca wskazanego przez wskaźnik wstatus wstawia “kod” zawierający informację o przyczynie zakończenia procesu (np. zwrócenie 1 z funkcji main)
  - Pamiętamy z Uniksa procesy w stanie zombie?
- pid_t waitpid(pid_t pid, int *wstatus, int options)
  - Pozwala na dokładniejsze określenie na jaki proces czekamy (dowolny, konkretny, jeden z grupy…)
  - Ma m.in. opcję pozwalającą na natychmiastowy powrót z funkcji, zamiast oczekiwania na potomków
- Obie funkcje opisane w man 2 wait

~~One~~ Three more things...

- int getrusage(int who, struct rusage *usage)
  - Przez wskaźnik rusage zwróci nam strukturę opisującą zużycie zasobów przez… who
  - Struktura rusage ma mnóstwo pól, częściowo “unmaintained” na Linuksie
- pid_t wait3(int *wstatus, int options, struct rusage *rusage)
  - Przez wskaźnik rusage zwróci nam strukturę opisującą zużycie zasobów przez potomka
  - Odpowiada waitpid(-1, wstatus, options) i ręcznemu wywołaniu getrusage
  - Czyli jeśli zignorujemy options, to odpowiada wait(wstatus)... :)
- pid_t wait4(pid_t pid, int *wstatus, int options, struct rusage *rusage)
  - Przez wskaźnik rusage zwróci nam strukturę opisującą zużycie zasobów przez potomka
  - Odpowiada waitpid(pid, wstatus, options) i ręcznemu wywołaniu gerusage
- Obie funkcje opisane w man 2 wait4

Struktura rusage

```
struct rusage {
    struct timeval ru_utime; /* user CPU time used */
    struct timeval ru_stime; /* system CPU time used */
    long   ru_maxrss;        /* maximum resident set size */
    long   ru_ixrss;         /* integral shared memory size */
    long   ru_idrss;         /* integral unshared data size */
    long   ru_isrss;         /* integral unshared stack size */
    long   ru_minflt;        /* page reclaims (soft page faults) */
    long   ru_majflt;        /* page faults (hard page faults) */
    long   ru_nswap;         /* swaps */
    long   ru_inblock;       /* block input operations */
    long   ru_oublock;       /* block output operations */
    long   ru_msgsnd;        /* IPC messages sent */
    long   ru_msgrcv;        /* IPC messages received */
    long   ru_nsignals;      /* signals received */
    long   ru_nvcsw;         /* voluntary context switches */
    long   ru_nivcsw;        /* involuntary context switches */
};
```

wstatus, wstatus… O co chodzi?

- WIFEXITED(wstatus) - zwraca prawdę, jeśli program “wyszedł” w wyniku zwrócenia wartości z funkcji main lub wywołania funkcji exit
- WEXITSTATUS(wstatus) - zwraca kod wyjścia programu (8 bitów!) dla którego WIFEXITED(wstatus) jest prawdziwe

- WIFSIGNALED(wstatus) - program umarł od sygnału
- WTERMSIG(wstatus) - informacja jakim sygnałem “oberwał”
- WIFSTOPPED(wstatus) - program został zatrzymany sygnałem
- WSTOPSIG(wstatus) - informacja jakim sygnałem został zatrzymany
- WIFCONTINUED(wstatus) - program został wznowiony sygnałem

Ograniczanie zasobów

- int getrlimit(int resource, struct rlimit *rlim)
  - Dla wybranego rodzaju zasobów zwraca do struktury wskazanej przez rlim aktualne miękkie i twarde ograniczenie zasobów
- int setrlimit(int resource, const struct rlimit *rlim)
  - Dla wybranego rodzaju zasobów próbuje ustawić limity zgodnie z wartościami podanymi w strukturze rlimit wskazywanej przez rlim
  - Ograniczenie twarde może być zmieniane przez użytkownika (nie-roota) wyłącznie na bardziej restrykcyjne; ograniczenie miękkie może być swobodnie zmieniane w dół i w górę, ale tylko do granicy wyznaczonej przez ograniczenie twarde
- Obie funkcje opisane w man 2 getrlimit

Rodzaje zasobów i struktura rlim

```
struct rlimit {
    rlim_t rlim_cur;  /* Soft limit */
    rlim_t rlim_max;  /* Hard limit (ceiling for rlim_cur) */
};
```

Wartość RLIM_INFINITY oznacza brak ograniczenia na dany zasób

- **RLIMIT_AS - ograniczenie wielkości pamięci wirtualnej (w bajtach)**
- **RLIMIT_CPU - ograniczenie zużytego czasu procesora**
  - Po przekroczeniu limitu miękkiego proces co sekundę dostaje sygnał SIGXCPU (żeby się pospieszył z tym co robi i już skończył)
- RLIMIT_FSIZE - wielkość utworzonych plików
- RLIMIT_LOCKS - ilość założonych locków
- RLIMIT_NOFILE - ilość otwartych deskryptorów pliku
- RLIMIT_STACK - limit na wielkość stosu (może być konieczna zmiana np. przy głębokiej rekurencji)

Uruchamianie plików wykonywalnych, czyli execve...

- int execve(const char *filename, char *const argv[], char *const envp[])
  - Dokumentacja dostępna w man 2 execve (opisane m.in. co nie zostaje zachowane z zasobów i środowiska procesu wołającego)
  - Jeśli się powiedzie, nowa binarka w całości zastępuje program w bieżącym procesie - funkcja nigdy nie zwróci nam wartości, jeśli zakończyła się powodzeniem
  - filename musi być pełną ścieżką do pliku wykonywalnego
  - argv[0] powinno być nazwą programu (przy typowym wywołaniu przez użytkownika mamy tam nazwę komendy lub ścieżkę względną/bezwzględną)
    - Wykorzystywane przez multi-call binaries - na dysku do tej samej binarki prowadzi wiele symlinków, a program orientuje się ”co ma robić” w oparciu o zawartość argv[0]
  - Kolejne elementy tablicy argv to kolejne argumenty z jakimi zostanie wywołany program
  - Kolejne elementy tablicy envp powinny mieć postać napisów "ZMIENNA=WARTOSC", jak w argumencie funkcji putenv czy w tablicy dostępnej w zmiennej environ
  - Ostatnim elementem tablic argv i envp powinien być NULL (sentinel/wartownik)

… i biblioteczni przyjaciele execve

- int execv(const char *path, char *const argv[])
- int execvp(const char *file, char *const argv[])
- int execvpe(const char *file, char *const argv[], char *const envp[])
- int execl(const char *path, const char *arg, … /* (char *) NULL */)
- int execlp(const char *file, const char *arg, ... /* (char *) NULL */)
- int execle(const char *path, const char *arg, ... /*, (char *) NULL, char * const envp[] */)

- Wszystkie funkcje opisane w dokumentacji man 3 exec

- Ale czym one wszystkie się różnią...?

… i biblioteczni przyjaciele execve

- Jeśli nazwa funkcji ma w końcówce e, to jako ostatni argument przyjmuje tablicę envp z listą zmiennych środowiskowych (jak w execve). Jeśli nie ma, to do uruchamianego programu przekazuje bieżące środowisko (environ).
  - execle, execvpe (i wywołanie systemowe execve)
- Jeśli nazwa funkcji ma w końcówce v, to argumenty wywołania programu przyjmuje jako pojedynczy argument argv, będący tablicą (jak w execve).
  - execv, execvp, execvpe (i wywołanie systemowe execve)
- Jeśli nazwa funkcji ma w końcówce l, to argumenty programu należy podać jako kolejne argumenty funkcji, a na końcu ich listy umieścić (char *) NULL.
  - execl, execle
- Jeśli nazwa funkcji ma w końcówce p, to pierwszy argument nie musi być kompletną ścieżką do programu - podana nazwa będzie wyszukiwana w katalogach podanych w zmiennej środowiskowej PATH
  - execlp, execvp, execvpe

Pod kątem zadania...

- Funkcja getline (man 3 getline) pozwala wczytać całą pojedynczą linię z pliku tekstowego i sama zajmuje się zaalokowaniem odpowiedniej ilości pamięci
- Funkcja strtok/strtok_r (man 3 strtok) pozwala znaleźć w tekście pozycje od których zaczynają się kolejne argumenty dla wywoływanych programów
- Można zrobić tablicę char** o stałej wielkości (obsługiwać np. max 5 argumentów) i pod kolejnymi indeksami podpinać kolejne wyniki strtok
- Jeśli w pliku wsadowym pojawi się "arg. ze spacją", to nie trzeba go specjalnie traktować - można rozbić go na spacjach: ["arg.], [ze], [spacją"]
- Żeby proces potomny mógł wypisywać komunikaty na ekran nie trzeba robić nic dodatkowo, wystarczy go po prostu uruchomić

Drobna inspiracja...

```
pid = fork();
if (pid == 0) {
    setrlimit(___);
    if (execvp(___) == -1) {
        perror("Karramba - execve");
        exit(1);
    }
}
else if (pid > 0) {
    wait3(___, &rusage);
    if (WIFEXITED(___) && WEXITSTATUS(___) != 0) {
        fprintf(stderr, "Porażka naszego potomka :(\n");
        exit(2);
    }
    else {
        printf("Zasoby: %d\n", rusage.___);
    }
}
else {
    perror("Ojej");
    exit(3);
}
```

Dziękuję za uwagę