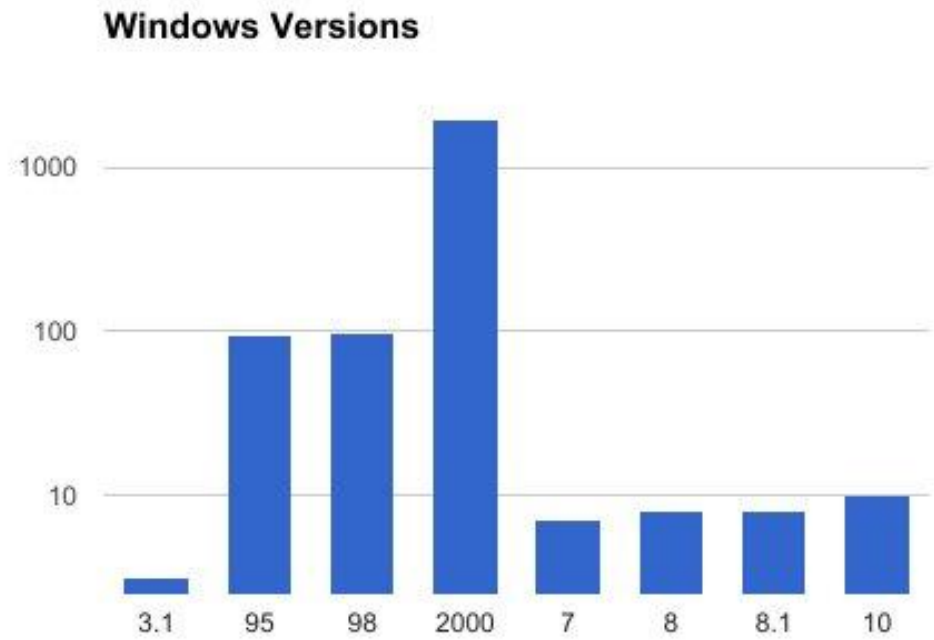
Systemy operacyjne
Lab 1

Mateusz Majcher

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
AGH University of Science and Technology

Kontakt

email: majcher@agh.edu.pl



Zasady zaliczenia

Wszystkie szczegóły w kursie “Systemy operacyjne 2020-2021” na UPeL

- Hasło:

Ocena z laboratorium jest średnią ważoną

- Zestawy zadań i aktywność (waga 0.7)
- Kolokwia (waga 0.3)

Oceny są wyznaczane zgodnie ze skalą ocen z regulaminu studiów na AGH

Zestawy zadań

Wszystkie zadania, które mają być zrealizowane na dany dzień, powinny zostać umieszczone w platformie Moodle do końca dnia(do 23:59), **poprzedzającego** dzień, w którym odbywają się zajęcia - czas na oddanie każdego zestawu to (zwykle) tydzień

Za pierwszy, **rozpoczęty** tydzień opóźnienia oddania zestawu zadań, od końcowej łącznej oceny z zestawów, odjęte zostanie 0.1 punktu. Jednocześnie, opóźnienie oddania zestawu **nie może przekroczyć** jednego tygodnia, gdyż wówczas skutkuje to oceną **niedostateczną** z laboratorium.

Pozytywne zaliczenie **wszystkich** zestawów zadań jest jednym z warunków niezbędnych do zaliczenia laboratorium.

Prowadzący będzie na zajęciach sprawdzał rozwiązanie zadań zaprezentowanych przez studenta oraz stopień opanowania obowiązującego materiału - 3-4 odpowiedzi ustne z prezentacją zadania

Podczas zajęć **może** odbyć się krótka kartkówka z materiału związanego z zestawem, który jest oddawany w danym dniu.

W ramach oceny składowej za realizację zestawów zadań prowadzący mogą oceniać przygotowanie do zajęć lub w wykazywaną aktywność.

Kolokwia

W ciągu całego semestru student ma obowiązek napisać **dwa** kolokwia.

Terminy kolokwiów zostaną podane przez prowadzącego ze stosownym wyprzedzeniem.

Uzyskanie pozytywnej średniej oceny ze **wszystkich** kolokwiów jest niezbędne do zaliczenia przedmiotu.

Praktyczne zadania.

Egzamin

Pisemny z wykładu (10~15 pytań)

Zerówka ustna - 5.0 z laboratoriów

Laboratorium 1

Opcje kompilatora gcc/g++. Narzędzia: GDB, Make

Zarządzanie pamięcią, biblioteki, pomiar czasu

Opcje kompilatora gcc/g++

-E - powoduje wygenerowanie kodu programu ze zmianami, wprowadzonymi przez preprocesor

- Wszystkie #include, #define i #if zostały rozwinięte w pełny kod

-S - zamiana kodu w języku C na kod asemblera

- Kod został przekształcony do postaci instrukcji asemblera (domyślnie do pliku *.s)
- Można tutaj obserwować działanie różnych optymalizacji

-c - kompilacja bez łączenia z bibliotekami

- Mamy gotowy kod maszynowy, ale program jeszcze nie nadaje się do uruchomienia (np. “obiecaliśmy” że funkcja x istnieje, ale jeszcze nie wiadomo gdzie ona jest)

Opcje kompilatora gcc/g++

Różne poziomy optymalizacji

- -O0 - wyłączenie optymalizowania kodu, przydatne przy debugowaniu
- -O lub -O1 - optymalizacja tylko w zakresie podstawowym, skraca to proces kompilacji programu;
- -O2 - zalecany poziom optymalizacji, wszystkie bezpieczne optymalizacje, tzn. nie zostanie włączona żadna optymalizacja, która mogłaby zmienić w istotny sposób działanie programu (np. zmniejszyć precyzję obliczeń zmiennoprzecinkowych)
- -Os - wariant -O2, w którym wyłączone są optymalizacje mogące zwiększyć rozmiar kodu - kod optymalizowany w celu minimalizacji rozmiaru pliku wykonywalnego
- -O3 - zawiera wszystko z -O2 + dodatkowe, agresywne optymalizacje, może powodować problemy w działaniu skomplikowanych programów, zmniejszyć precyzję obliczeń i spowodować znaczny wzrost objętości kodu maszynowego programu
- -Ofast - wariant -O3, w którym aktywowane są dodatkowe optymalizacje mogące uczynić kod niezgodnym ze standardem języka

Wykorzystanie różnych zestawów instrukcji procesora

-march=... - nazwa konkretnej rodziny procesorów lub opcje takie jak generic (“uniwersalny”) lub native (dokładnie pod ten komputer, na którym kompilujemy źródła)

Opcje kompilatora `gcc/g++`

Opcje linkera:

- -lbiblioteka - (mała litera l) wymusza łączenie programu z podaną biblioteką = dołącza skompilowaną bibliotekę
- -L./sciezka - pokazuje katalog gdzie są pliki biblioteki
- -shared - służy do skompilowania biblioteki dzielonej
- -static - służy do zlinkowania programu i bibliotek w sposób statyczny

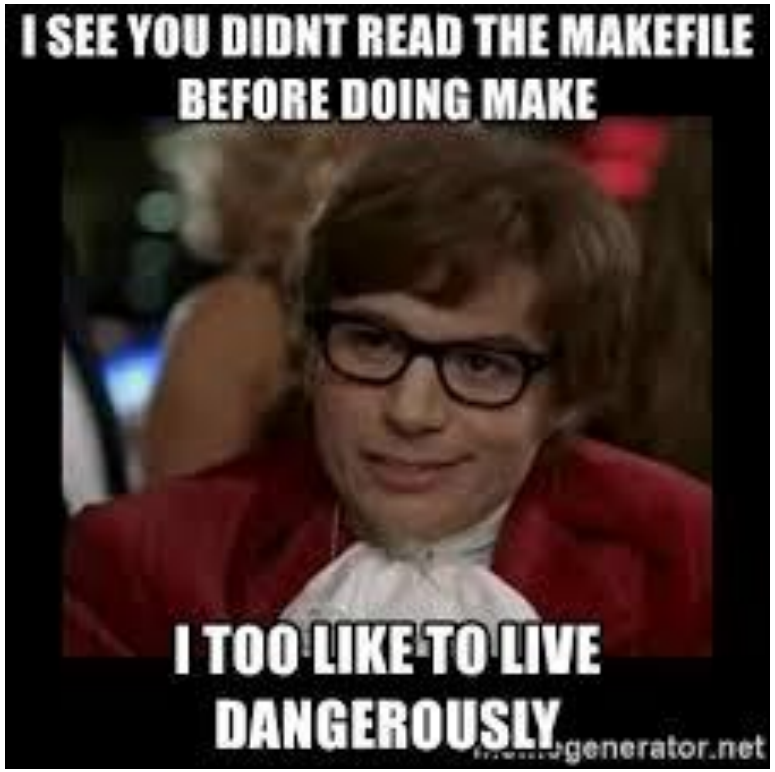
Inne przydatne opcje:

- -DNAZWA=WARTOSC - ma efekt identyczny jak umieszczenie w kodzie linijki #define NAZWA WARTOSC
- -g - umieszcza w programie wynikowym informacje pozwalające na jego skuteczne debugowanie (np. przy użyciu GDB)
- -fPIC - Position Independent Code, potrzebne przy bibiliotekach
- -Wall - włącza wszystkie możliwe ostrzeżenia w trakcie kompilacji
- -std=... - pozwala podać wersję standardu języka, której chcemy używać (np. -std=c11 dla standardu języka C z roku 2011)
- -pthread - włączenie obsługi POSIXowych wątków

Narzędzia: `Make`

- Program make jest częścią projektu GNU.
- Został stworzony na potrzeby automatyzacji procesu kompilacji oraz budowania projektu. Jest on interpreterem plików nazywanych Makefile'ami, w których możemy tworzyć własne reguły kompilacji w taki sposób, aby można było za pomocą dwóch-trzech komend zamienić kod źródłowy w pliki wykonywalne, biblioteki itp.

- Tworzymy plik tekstowy o specjalnej strukturze z opisem budowy różnych elementów programu
- Taki plik nazywamy Makefile
- Po wywołaniu komendy make plik zostaje przetworzony, a zażądana przez użytkownika operacja - wykonana



Narzędzia: Make

Reguły postaci:

cele … : zależności …
    polecenia

Zmienne specjalne:

- CC - kompilator C
- CFLAGS - flagi kompilacji C
- CPPFLAGS - flagi preprocesora C
- LDFLAGS - flagi linkera
- LDLIBS - nazwy bibliotek do zlinkowania z programem

```
# Makefile

main: clean
    gcc main.cpp -o main

clean:
    rm -f main
```

```
# Makefile
CC = gcc
CONFIG_DIR?="/etc/myprogram"

main: clean
    $(CC) main.cpp -o main

clean:
    rm -f main

configure:
    echo "${CONFIG_DIR}"
```

Narzędzie GDB

- Debugger - program służący do analizy przebiegu wykonania programu
- Pozwala na:
  - Uruchomienie nowego procesu lub podłączenie się do istniejącego i śledzenie ich wykonania
  - Analizę “post mortem” przyczyn awarii programu w sytuacji kiedy dostępny jest jego zrzut
  - pamięci z momentu zdarzenia (“core dump”)

- Podczas kompilacji aplikacji mogą zostać wygenerowane symbole, czyli dodatkowe informacje przeznaczone dla debuggera. Mogą to być między innymi dane na temat zmiennych, funkcji, odpowiednich numerów linii w pliku źródłowym, czy użytym języku programowania.
- Na ich podstawie debugger pozwala odwoływać się do poszczególnych elementów programu w przystępny sposób. By dołączyć odpowiednie symbole debugowania, wystarczy użyć przełącznika **-g** podawanego kompilatorowi.

Narzędzie GDB

Jak rozpocząć pracę z GDB?

- Uruchomienie programu
  pod kontrolą GDB:
  $ gdb ./example.out
  Reading symbols from ./example
  (gdb) run

- Podłączenie się do już uruchomionego procesu danego programu:
  $ gdb ./example.out
  (gdb) attach 1593

- Zamknięcie GDB następuje po wykonaniu polecenia quit:
  $ gdb ./example.out
  (gdb) quit

Narzędzie GDB

- **run** - uruchomienie aplikacji
- **break N** - ustawienie breakpointa w linii N
- **disable/enable X** - włączenie lub wyłączenie breakpointa X
- **watch X** - zatrzymywanie programu przy zmianach wartości zmiennej X
- **step** - wykonanie pojedynczego kroku po zatrzymaniu programu na breakpoince
- **finish** - przeskoczenie do końca pętli, funkcji
- **continue** - kontynuowanie wykonania programu do następnego breakpointa
- **set variable X = WARTOSC** - zmiana wartości zmiennej X na wartość

- **print X** - wyświetlenie wartości zmiennej X
- **info locals** - wyświetlenie informacji o wartościach zmiennych lokalnych
- **info args** - wyświetlenie informacji o argumentach funkcji
- **info break** - wyświetlenie informacji o aktualnie ustawionych breakpointach
- **info sharedlibrary** - wyświetlenie informacji o załadowanych bibliotekach dzielonych działających wątkach
- **backtrace; bt** - wyświetlenie jak znaleźliśmy się w danym miejscu programu (ramki stosu)
- **nexti** - wykonaj jedną instrukcję i wstrzymaj debugowanie

Biblioteki

- Żeby korzystanie z bibliotek miało sens, należy oddzielić deklaracje funkcji od ich definicji
  - Deklaracje funkcji (nazwa, argumenty, zwraca wartość) i struktur - w pliku nagłówkowym *.h
  - Definicje (kod implementacji) - w pliku *.c biblioteki
- W kodzie który będzie korzystał z biblioteki include'ujemy plik nagłówkowy *.h
- Plik nagłówkowy powinien zawierać tzw. "header guard" - żeby uniknąć błędów "already declared":

```
#ifndef _NAGLOWEK_H
#define _NAGLOWEK_H
// tutaj deklaracje
#endif
```

- Jeśli bibliotekę tworzymy w C++ i chcemy z niej korzystać w C, musimy deklaracje umieścić w bloku extern "C":

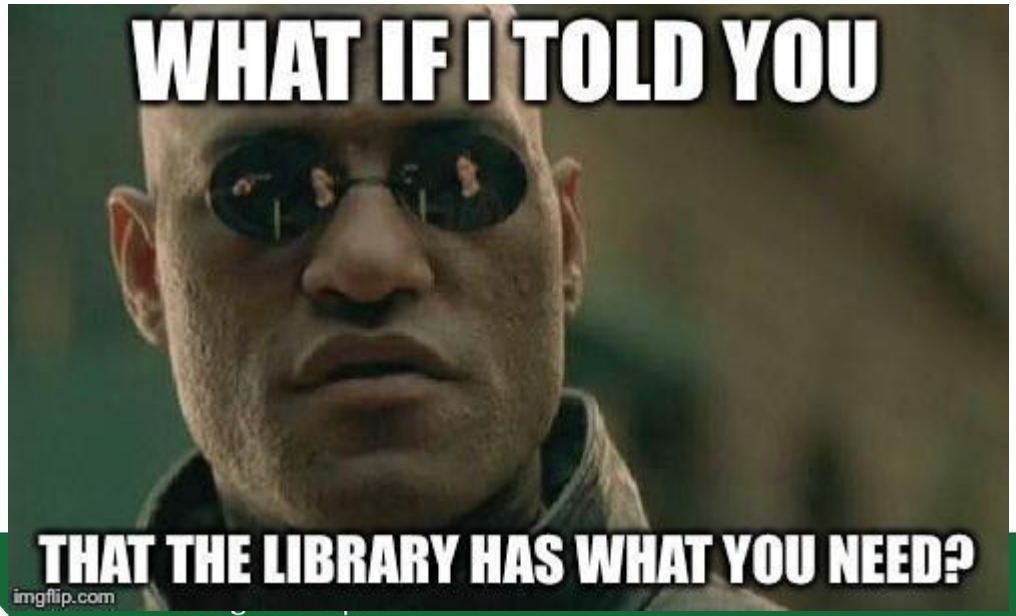
```
extern "C" {
// tutaj deklaracje
}
```

- Jest to związane ze sposobem w jaki linker "radzi sobie" z przeciążaniem funkcji w C++ - symbol funkcji musi być unikatowy, w związku z czym nazwy funkcji zostają przekształcone tak, by zawierały pewną informację o ich argumentach.

Biblioteki

Biblioteka statyczna:

- Zwykle w postaci archiwum *.a
- Archiwum zawiera zasemblowane pliki *.o
- Dołączenie do programu biblioteki statycznej powoduje włączenie jej skompilowanego i zasemblowanego kodu w skład programu
- Program zlinkowany z biblioteką statycznie zajmuje więcej miejsca na dysku, ale nie wymaga dostarczania tej biblioteki dodatkowo (bo zawiera ją “w sobie”)



Biblioteka dzielona/ładowana dynamicznie:

- W postaci pliku *.so
- Można dołączyć ją do programu na dwa
- sposoby:
  - W trakcie linkowania - w programie zostaną umieszczone wskazania na symbole pochodzące z biblioteki (zamiast jej kodu)
  - W trakcie działania programu - odwoływać się do niej z wykorzystaniem funkcji z nagłówka dlfcn.h i biblioteki dl
- Jeśli z biblioteki korzysta wiele programów, oszczędzamy miejsce i ułatwiamy sobie jej aktualizowanie
- Potencjalny koszt: zaktualizowana biblioteka przestaje być zgodna z programami które z niej korzystały

Biblioteki statyczne

- Tworzymy zasemblowany kod biblioteki:
  - gcc -c biblioteka.c -o biblioteka.o
- Pakujemy bibliotekę do archiwum:
  - ar rcs libbiblioteka.a biblioteka.o
- W trakcie kompilacji naszego programu:
  - gcc program.c -L. -lbiblioteka -o program
    - -L. - oznacza szukanie biblioteki/archiwum z nią w bieżącym katalogu
    - przedrostek lib i rozszerzenie .a (lub .so) zostaną automatycznie uwzględnione w trakcie poszukiwań biblioteki
- Jeśli istnieje zarówno statyczna jak i dzielona wersja biblioteki, to domyślnie zostanie użyta biblioteka dzielona; możemy wymusić statyczne linkowanie całego programu:
  - gcc program.c -static -L. -lbiblioteka -o program
    - Żeby taka kompilacja się powiodła, w systemie musi być obecna również statyczna wersja biblioteki standardowej C!

Biblioteki dzielone

- Tworzymy bibliotekę:
  - gcc -fPIC -shared biblioteka.c -o libbiblioteka.so (możemy też użyć biblioteka.o zamiast biblioteka.c)
    - -fPIC - w różnych programach biblioteka może trafić pod różne adresy w pamięci, więc instrukcje skoku muszą być względne, a nie do konkretnych adresów (Position Independent Code)
- W trakcie kompilacji naszego programu:
  - gcc program.c -L. -Wl,-rpath=. -lbiblioteka -o program
    - -L. - oznacza szukanie biblioteki w bieżącym katalogu na etapie linkowania (w celu odczytania tablicy symboli)
    - -Wl,-rpath=. - oznacza szukanie biblioteki w bieżącym katalogu w momencie uruchomienia programu
- Użycie -Wl,-rpath=. przydaje się raczej w eksperymentach; na co dzień zwykle stosuje się jedno z następujących podejść:
  - umieszczenie biblioteki w odpowiednich lokalizacjach systemowych (np. /usr/lib64)
  - dodanie katalogu z biblioteką do konfiguracji dynamicznego loadera (/etc/ld.so.conf) i wywołanie ldconfig
  - wskazanie w zmiennej środowiskowej LD_LIBRARY_PATH dodatkowych katalogów do uwzględnienia w trakcie poszukiwań biblioteki
- Komenda ldd pozwala nam podejrzeć z jakimi bibliotekami został dynamicznie zlinkowany nasz program

Biblioteki dzielone ładowane przez program

- Korzystamy z tego samego pliku *.so z biblioteką co w przypadku linkowania dynamicznego
- W trakcie kompilacji naszego programu nie odwołujemy się do naszej biblioteki; zamiast tego dołączamy bibliotekę dynamicznego loadera (dl):
  - gcc program.c -ldl -o program
- Zmienia się konstrukcja naszego programu:
  - Dołączamy plik nagłówkowy dlfcn.h
  - Otwieramy plik z biblioteką funkcją void *dlopen(const char *filename, int flags)
  - Pozyskujemy z otwartej biblioteki uchwyt do funkcji z użyciem void *dlsym(void *handle, const char *symbol)
    - Konieczne wykonanie akrobacji z ustawieniem prawidłowego typu zmiennej-wskaźnika do funkcji
  - Wywołujemy funkcję, korzystając z zapisanego chwilę wcześniej wskaźnika
  - Zamykamy bibliotekę funkcją int dlclose(void *handle)
  - Manual: man 3 dlsym

Zarządzanie pamięcią

- nagłówek stdlib.h
- void *malloc(size_t size)
  - Zaalokowanie size bajtów pamięci bez jej inicjalizowania
  - Uwaga na overflow iloczynu jeśli robimy coś w stylu obszar = malloc(nmemb * size)
- void *calloc(size_t nmemb, size_t size)
  - Zaalokowanie nmemb * size (bez obaw o overflow) bajtów pamięci + zainicjalizowanie zerami
  - Prawie jak malloc + memset..
- void *realloc(void *ptr, size_t size)
  - Zmiana rozmiaru zaalokowanego dynamicznie obszaru pamięci
  - Jeśli nie dało się po prostu go rozszerzyć, przekopiuje dane w nowe miejsce i zwolni stary

My C program exit
without freeing allocated memory

OS:

- void free(void *ptr)
  - Zwolnienie zaalokowanego dynamicznie obszaru pamięci
  - Próba ponownego zwolnienia już zwolnionego obszaru jest błędem!
- Manual: man 3 malloc

Pomiar czasu

- Zewnętrznie (w środowisku):
  - Wbudowana funkcja powłoki bash:

    [mkwm:~] $ time /usr/bin/sleep 3
    real 0m3.004s
    user 0m0.000s
    sys 0m0.003s

  - Polecenie time (wymagana podanie pełnej ścieżki, żeby nie wykorzystać funkcji powłoki!):

    [mkwm:~] $ /usr/bin/time /usr/bin/sleep 3
    0.00user 0.00system 0:03.00elapsed 0%CPU (0avgtext+0avgdata 2020maxresident)k
    0inputs+0outputs (0major+66minor)pagefaults 0swaps

  - Manual: man 1 time (zawiera m.in. sposób zmiany formatu wyświetlania wyniku)

Pomiar czasu

- Wewnętrznie (w programie):
  - clock_t times(struct tms *buf) z sys/times.h
    - Przez podany wskaźnik aktualizuje strukturę ze zużyciem czasu procesora i użytkownika w “tickach”
    - Ilość “tików” w sekundzie zwróci wywołanie: sysconf(_SC_CLK_TCK);
    - “To measure changes in elapsed time, use clock_gettime(2) instead” - co pewien czas jest overflow
    - Manual: man 2 times
  - int clock_gettime(clockid_t clk_id, struct timespec *tp) z time.h
    - Interesujące z naszego punktu widzenia są zegary CLOCK_REALTIME i/lub CLOCK_MONOTONIC
    - Przez podany wskaźnik aktualizuje strukturę z sekundami i nanosekundami dla wybranego zegara
    - Manual: man 3 clock_gettime
  - int getrusage(int who, struct rusage *usage) z sys/resource.h
    - Pozwala odczytać zużycie zasobów dla “wołającego” procesu lub wątku oraz dla procesów potomnych
    - Przez podany wskaźnik aktualizuje strukturę która zawiera dużo więcej niż tylko czas procesora i czas użytkownika
    - Manual: man 2 get_rusage