Debugowanie

Do kompilatora należy dodać odpowiednią opcję, która do skompilowanego programu dołączy informacje służące do debugowania:

-g, lub ***-ggdb***, bądź ***-ggdb3***.

Flagę należy dodać do pliku cmake w opcjach kompilatora (***add_compile_options()***).

Po dodaniu flagi możemy debugować oprogramowanie za pomocą interface'u graficznego Visual Studio Code.

Debugowanie za pomoca gdb

Debugowanie możemy przeprowadzić za pomocą konsoli przy użyciu narzędzia GDB.
gdb <executable file> - opcjonalnie przekazujemy plik do debugowania

Komenda gdb otwiera shell debuggera. Po programie poruszamy się poniższymi komendami:

(gdb) ***file <executable file>*** - wskazuje plik do debugowania
(gdb) ***break <source_file>:<line_number>*** - dodaje punkt przerwania we wskazanej linii
(gdb) ***break <function name>*** - dodaje punkt przerwania we wskazanej funkcji
(gdb) ***run*** - uruchamia program
(gdb) ***continue*** - kontynuowanie programu do kolejnego punktu przerwania
(gdb) ***step*** - wykonanie kolejnej instrukcji
(gdb) ***next*** - wykonanie kolejnej linii
(gdb) ***print <variable>*** - wypisanej wartości zmiennej <*variable*>
(gdb) ***watch <variable>*** - dodanie zmiennej <*variable*> do listy obserwowanych
(gdb) ***bt [[-]n] [full]*** - wypisanie stosu wywołań do n elementów stosu
(gdb) **frame <N>** - wybór obserwowanej ramki w stosie wywołań
(gdb) ***q*** - wyjście z *gdb*.

Analiza dynamiczna programu

Do dynamicznej analizy oprogramowania posłuży nam oprogramowanie Valgrind. Jest to zestaw narzędzi które pozwala na wykrywanie błędów związanych z zarządzaniem pamięcią i wątkami. Ponadto pozwala także na profilowanie.

Valgrind domyślnie wykorzystuje narzędzie ***memcheck***, które analizuje program pod kątem operacji na pamięci, tj. wyciek, odczyt i zapis.

valgrind
--tool=memcheck - zbędne, wybór domyślnego narzędzia *memcheck*
--log-file=valgrind-out.txt - opcjonalny zapis logów do pliku
./<exec_file> <params> - plik wykonywalny z parametrami

W Valgrind możemy uzyskać szczegóły na temat wykrytych błędów operacji na pamięci, do tego służą dodatkowe opcje narzędzia.

valgrind
--leak-check=full - wyświetla szczegóły o wykrytych wyciekach pamięci
--verbose - wyświetla informacje o nietypowym zachowaniu
--log-file=valgrind-out.txt - opcjonalny zapis logów do pliku
./<exec_file> <params> - plik wykonywalny z parametrami

Wykryte błędy są bardziej czytelne dla programu bez symboli służących do debugowania.

Profilowanie oprogramowania

Valgrind zawiera także narzędzie Callgrind, służące do profilowania kodu. Callgrind generuje stos wywołań wskazanego programu i zapisuje go do pliku, domyślnie o nazwie *callgrind.out.XXXXX*, gdzie *XXXXX* to PID programu.

valgrind
--tool=callgrind - wybór Callgrind'a jako używanego narzędzia
--callgrind-out-file=<file> - opcjonalne wskazanie pliku wyjściowego
./<exec_file> <params> - plik wykonywalny z parametrami

Aby uzyskać poprawne pomiary wydajności kodu należy analizować kod bez informacji służących do debugowania.

Graficzna wizualizacja profilowania

Do wizualizacji stosu wywołań oprogramowania wygenerowanego przy użyciu Callgrind'a możemy wykorzystać narzędzie KCacheGrind.

kcachegrind
./<callgrind_out_file> - plik wygenerowany przez narzędzie Callgrind