5.2.2 Identificando os defeitos

Conforme apresentado no relatório de AUTETESE, a aplicação *thread_test* falhou em todas execuções, independente das configurações apresentadas. Desta forma, a entrega do trabalho não é satisfatória e o grupo responsável deve descobrir o defeito inserido e corrigir a implementação.

A aplicação de testes das *threads* foi desenvolvida para verificar se as operações comuns a este tipo de abstração estão corretamente implementadas. A Figura 23 ilustra o trecho de código da aplicação, a qual inicia-se com a *thread m* criando mais duas *threads* (*a* e *b*), depois *m* espera que ambas finalizem e exclui todas as *threads*, realizando inclusive uma auto deleção.

```
14  Thread * a;
15  Thread * b;
16  Thread * m;
17
18  OStream cout;
19
20  int main()
21  {
22      cout << "Thread test" << endl;
23
24      m = Thread::self();
25
26      cout << "I'm the first thread of the first task created in the system." << endl;
27      cout << "I'll now create two threads and then wait for them to finish ..." << endl;
28
29      a = new Thread(&func_a);
30      b = new Thread(&func_b);
31
32      int status_a = a->join();
33      int status_b = b->join();
34
35      cout << "Thread A exited with status " << status_a
36           << " and thread B exited with status " << status_b << "" << endl;
37
38      delete a;
39      delete b;
40      delete m;
41
42      cout << "It should not be shown on the display!" << endl;
43
44      return 0;
45  }
```

Figura 23 – Trecho de código da aplicação *thread_test*

Na Figura 24 encontra-se o resultado da execução da aplicação, que continuou mesmo depois da *thread m* executar o comando de auto deleção. AUTETESE revelou que esta aplicação possui o processamento incorreto, pois no arquivo de *log* da execução não poderia aparecer a frase *"It should not be shown on the display!"*.

Quando a aplicação é reprovada, uma instância do GDB é automaticamente é disponibilizada para que o usuário possa encontrar o erro. Para esta aplicação foram inseridos *breakpoints* e *watchpoints* conforme a Figura 25.

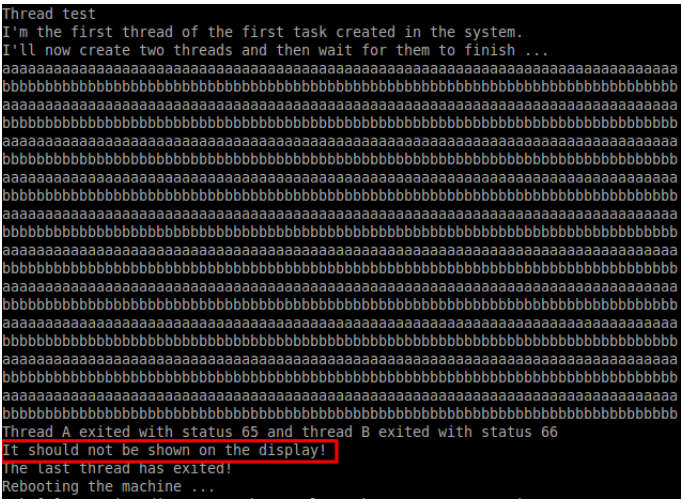

Figura 24 – Resultado da execução falha da aplicação *thread_test*.

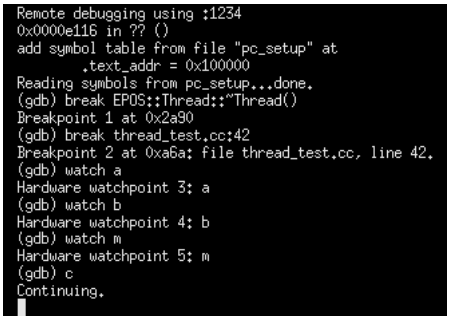

Figura 25 – Depuração da aplicação *thread_test*.

A depuração ocorreu no modo *step* e foi possível verificar que chamada do método de destrução das *threads* foi executado para todas as *threads*, entretanto, apenas *a* e *b* foram realmente deletadas. Analisando o código fonte, notou-se que a *thread m* não entrou em nenhuma das condições da destrução. Analisando a implementação do destrutor da *thread*, descobriu-se que o erro ocorreu no tratamento do término da última *thread*, que não considerou o caso de haver uma *thread* esperando na fila de um sincronizador.

Para resolver este problema, os alunos responsáveis optaram dor desenvolver uma lista de recursos que cada abstração do sistema deve gerenciar. Sendo assim, quando a *thread* for deletada é necessário liberar todos os sincronizadores em que ela possa estar. Além disso, agora a *thread* não pode de auto deletar. O novo trecho de código fonte está ilustrado na Figura 26.

```
Thread::~Thread()
{
    lock();

    assert(_state != RUNNING);

    switch(_state) {
    case RUNNING:
        exit(-1);
        break;
    case READY:
        _ready.remove(this);
        _thread_count--;
        break;
    case SUSPENDED:
        _suspended.remove(this);
        _thread_count--;
        break;
    case WAITING:
        _waiting->remove(this);
        _thread_count--;
        break;
    case FINISHING:
        break;
    }

    if(_joining)
        _joining->resume();

    unlock();

    kfree(_stack);
}
```

Figura 26 – Código fonte do destrutor da *thread* após correção.

Após a correção, AUTETESE foi novamente e o resultado foi positivo para a aplicação *thread_test*. Este defeito na auto deleção não foi exclusivo de um grupo, pois mais de 25% da turma não se atentou ao tratamento desta situação e não realizou a entrega corretamente.