Part 1 introduced the partially documented initialization parameter `EVENT` and some of the benefits that may be realized by using it for troubleshooting. Part 3 expands on that material in the sense that the events it discusses may be set using the parameter `EVENT` or the SQL statements `ALTER SESSION`/`SYSTEM`. The difference between these approaches is that solely the parameter `EVENT` ensures that the configuration is both persistent and pertains to the entire lifetime of an ORACLE DBMS instance. The subsequent chapters address events for deadlock diagnosis, collection of performance data, and Oracle Net packet dumps.

**Deadlocks**

A deadlock occurs when two or more sessions hold locks and another lock request, which would result in a circular chain of locks, is issued. If this new lock request were granted, the sessions would deadlock and none of them would ever finish. Hence, the ORACLE DBMS detects circular chains pertaining to interdependent locks, signals the error “ORA-00060: deadlock detected while waiting for resource”, and rolls back one of the sessions involved in the would-be deadlock. A trace file is written whenever an ORACLE instance detects a deadlock. The undocumented event 10027 gives the DBA control over the amount and type of diagnostic information generated.

Figure 5-1 depicts a deadlock situation among two database sessions. Session 1 locks the row with `EMPLOYEE_ID=182` at time $t_1$. At time $t_2$, session 2 locks the row with `EMPLOYEE_ID=193`. At $t_3$, session 1 requests a lock on the row with `EMPLOYEE_ID=193`, which is already locked by session 2. Hence, session 1 has to wait on the event `enq: TX - row lock contention`. At $t_4$, session 2 requests a lock on the row that session 1 locked at $t_1$. Since granting this lock would lead to a circular chain, the DBMS signals “ORA-00060: deadlock detected while waiting for resource” at $t_5$. The `UPDATE` statement executed by session 1 at $t_3$ is rolled back. At this point, session 2 is still waiting for the lock on the row with `EMPLOYEE_ID=182`, which session 1 continues to hold. Session 1 should `ROLLBACK` in response to ORA-00060, releasing all its locks and allowing session 2 to complete the update of employee 182.
To avoid deadlocks, rows need to be locked in the same order by all database sessions. If this is not feasible, deadlocks and the overhead associated with writing trace files may be avoided by executing `SELECT FOR UPDATE NOWAIT` prior to an `UPDATE`. If this returns ORA-00054, then the session needs to roll back and reattempt the entire transaction. The `ROLLBACK` will allow other transactions to complete. The downside of this approach is the additional processing. Following is an example that is tailored to the previous scenario:

```sql
SQL> SELECT rowid FROM hr.employees WHERE employee_id=182 FOR UPDATE NOWAIT;
SELECT rowid FROM hr.employees WHERE employee_id=182 FOR UPDATE NOWAIT
* 
ERROR at line 1: 
ORA-00054: resource busy and acquire with NOWAIT specified
SQL> ROLLBACK;
Rollback complete.
SQL> SELECT rowid FROM hr.employees WHERE employee_id=182 FOR UPDATE NOWAIT;
ROWID

AADNLAAEAAAABb
SQL> UPDATE hr.employees SET phone_number='650.507.9878'
WHERE rowid='AADNLAAEAAAABb';
1 row updated.
```

### Figure 5-1. Deadlock detection

![Figure 5-1](image)

Event 10027 gives the DBA control over the amount and type of diagnostic information generated in response to ORA-00060. At default settings, an ORA-00060 trace file contains cached cursors, a deadlock graph, process state, current SQL statements of the sessions involved, and session wait history (in Oracle10g and subsequent releases). Except for the current SQL statements and the deadlock graph, all the information pertains merely to the session that received ORA-00060. Event 10027 may be used to achieve the following oppositional goals: